Testing of Embedded Software

Tessy 2.6

User Manual

Imprint

Tessy v2.6 for Windows 2000/XP/Vista

Sales

Razorcat Development GmbH

Witzlebenplatz 4

D-14057 Berlin

Tel: +49 (030) 53 63 57 0

Fax: +49 (030) 53 63 57 60

e-Mail: support@razorcat.com

Internet: http://www.razorcat.com/

Berlin, November 2008 :: *DocRev. 1.1.4*

Windows is a registered trademark of Microsoft. Tessy and CTE are registered trademarks of Razorcat Development GmbH.

All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same is claimed.

Author: flb, mw

© Razorcat Development GmbH 2008

This manual was produced using *ComponentOne Doc-To-Help.*™

Contents

Imprint ii
Contents iii

Preface **1**

About Tessy 1
Unit Testing with the Test System Tessy 1
Classification Tree Method and Classification Tree Editor 3
Automated Regression Testing 4
History 5
New Features in Version 2.6 6
Enhanced Coverage Visualization and Reporting 6
Automatic Test Data Generation 7
Graphical Reporting of Variable Values 7
C++ Support 7
Using Symbolic Constants as Test Data 7
Bitcheck 8
Value for the Return Value of Functions in the CTE 8
Interface Database Optimization 8
Tool Qualification Package 8
Support For Windows Vista 9
New Compiler and Debugger Integrations 9
Conventions 9

Installation **11**

Preparing for the Installation 11

Pre-Installation Checklist 12
Installing Tessy 14
Setup Overview 14
New Installation 15
Using Cygwin Versions other than Cygwin 1.5.18 29
Update a Previous Installed Version 31
Multiple Installations of Tessy 2.6.x 32
Modify, Repair or Remove Installation 35
Modify, Repair or Remove Installation 35
Registration 36
Floating License 36
Node-Locked License 37
Request License 37
Register Tessy 40
Install Local License Server 40
Install Central License Server 43
Using Floating License Manager 46
Checkout Licenses 49

Quickstart **53**

Tutorial 53
Start Tessy and Create a New Database 54
Create a Project, a Module 57
Specify Target Environment 59
Add Source File 60
Open the Module 61
Edit Test Object Interface 63
Enter Input and Expected Values 66
Execute Test Run 71
Evaluate Test 73

Test Organization **79**

Databases 79
The Project Root 79
Creating a Project Database 81
Open a Database 85
Close a Database 90
Delete a Database 90

Creating a Project 91
Add a Project 91
Open a Project 92
Rename a Project 92
Delete a Project 93
Creating a Module 93
Add a Module 94
Rename a Module 94
Copy and Paste Modules 95
Save a Module 95
Restore a Module 97
Reset a Module 97
Delete a Module 98
Specifying the Test Environment 98
Module Properties 99
General Tab 99
Sources Tab 101
Linker / Binary Tab 109
Attributes Tab 111
Comment Tab 114
Test Object Properties 115
General Tab 115
Attributes 115
Comment 117
Common Module Options 117
Check Source Files 117
Edit Source Files 117
Source Files Changed 118
Changed Interface 119
Interface Analysis 119

Tessy Environment Editor TEE 121

Configuration Management 121
Starting the TEE 122
TEE Structure 124
TEE and User Access Rights 126
TEE Configuration Files 126
Using Default Configurations 128
Advanced Configurations 133

Copying Configurations 134
Creating Configuration Files 137
Create a New Custom Configuration File 137
Open an Existing File 143
Adding / Editing Attributes 143
Edit Attribute 144
Add Attribute 146
Symbol Appearance 148
Representation of Values in TEE 150

Test Preparation 151

The Test Interface Editor TIE 151
Open the TIE 152
Browse the Interface 154
Set Passing Directions 157
Reset Passing Direction to the Initial Value 159
Passing Direction of Special Data Types 161
Define Stubs for Functions 163
Define or Declare External Variables 173
Define Missing Linker References 175
Search for Variables 175
Save Interface Information and Exit the TIE 176

Edit Test Cases 177

Enter Test Cases 177
Systematic Test Definition 178
Open the CTE 178
Assign Variables 180
Assign Variables 180
Assign Values to Variables 181
Export Test Items to Tessy 183
Synchronize CTE and Tessy 184
Representation of Test Results 185
Insert Test Cases Manually 186
Delete Test Cases 187
Delete Test Steps 188
Enter Test Case Description and Specification 189
Usercode 190

Starting the Usercode Editor UCE 191
Using the UCE 192
Enter C Code 193
Enter Script Commands 210
Close Usercode Window 211
Import/Export 211

Enter Test Data 215

The Test Data Editor TDE 215
Start the TDE 215
Structure of the TDE 217
Browse the Interface 218
Test Steps 222
Edit Input and Expected Values 227
Enter Values 230
Automatic Test Data Generation 244
Enter Evaluation Modes 247
Ignore Values for a Test Step 252
View Results after Test Execution 253
Adopt Test Results as Expected Values 254
Copy and Paste Test Data 255
Search for Variables 255
Exit the TDE 256

Running a Test 257

Steps to Perform 257
Structure of the Test Driver 258
Setting Test Options 259
Generating the Test Driver 259
Running Test Cases Separately 260
Coverage Analysis 261
Setting Breakpoints 262
Run Test with Current Options 263
Execute Test Run 263
Symbol Appearances 265
Change of the Test Case Status 267
Batch Test 268
Test Objects Tab 269

Options Tab 269
Test Report Tab 270
Save and Execute Batch Test 271
View Batch Test Report 273
Running Batch Tests at Command Line 275

Test Report 277

Creating Reports 277
Generating a Report after a Test Run 278
Test Report Structure 282
Test Reports with Results Plots 289
Test Reports with Timing Measurements 291

Monitoring of C1 and C2 293

View Coverage Results 293
Starting the Coverage Viewer CV 293
Window Structure of the CV 294
Flow Chart View 295
Displayed Elements 295
Searching for Uncovered Conditions or Unreached Branches 297
C1 Coverage View 300
Summarized Total Coverage 301
Branch Coverage for selected Test Steps 301
MC/DC Coverage View 301
Selecting Decisions 303
Uncovered Condition Combinations 305
MCC Coverage View 305
The Code View 305
Highlighting Code Fragments 305
Coverage Statistic View 307
Coverage Report Views 307

Regression Testing 309

Performing Regression Testing 309
The Interface Data Assign Editor IDA 309
Assigning Test Objects 310

Assigning Called Functions 314
Assigning Interfaces 315
Specify Options for Automatic Assignment 318
Accepting Changes and Exiting IDA 320

Traceability of Test Changes 323

Diff Viewer 323
Automated Collecting of Test Results 324
Compare Test Results 325
Create Snapshots 328
Clean History 328
Compare XML Reports 329

Troubleshooting 331

Contact Support 331
Enable Logging for Troubleshooting 332
Open Support Dialog 333
Error Messages 334
How to Handle Errors Appearing in the Message Window 334
Common Error Messages 335
Tessy Error Codes 335
FAQ 338
Errors While Opening a Module 338
Errors While Generating the Test Driver 339
Module Interface has Changed 339
Test Cases Can’t be Inserted 339
I Can't Choose a Certain Compiler or Debugger 340
Application Notes of Tessy 340

Technical Information 341

Overview 341
Compiler Settings 342
Debugger/Emulator Settings 342
Version Control 343
Files from the Module Folder 343
Files outside the Module Folder 344

Files in the Module Directory 344
Makefile 344
Generated Files 345
Technical Restrictions 347
Number of Test Cases and Test Steps 347
Tasking Compiler bit Variables 347
Test Object Interface Size 347
Structs/Unions/Enums and Multiple Source Files 347

Glossary of Terms **349**

Index **355**

Index 355

Preface

About Tessy

Dynamic testing is indispensable when testing a software system. Today, up to 80% of the development time and costs go into unit and integration testing. It is therefore of urgent necessity to automate testing processes in order to minimize required time and costs for developing high-quality products.

This is where the test system **Tessy** comes in: by automating the whole test cycle, unit testing for programs in C are optimally supported in all test phases. The system also takes care of the complete test organization as well as test management.

Unit Testing with the Test System Tessy

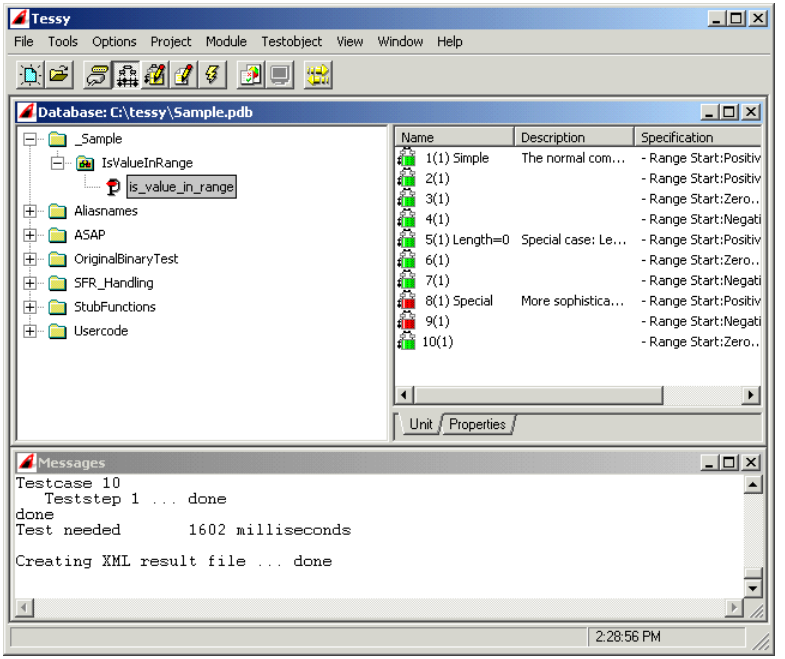
Tessy offers an integrated graphic user interface conducting you comfortably through the unit test. There are special tools for every testing activity as well as for all organizational and management tasks. The unit test in **Tessy** is divided into the following central test activities:

- Test case determination
- Test data and expected value determination
- Test execution

- Test evaluation and test documentation

Each phase is separately executed for every test object. Test objects are the export functions of a module.

The test organization takes place in projects containing modules to which, in turn, the respective test objects are assigned. The source code of a module is parsed automatically to determine the test objects and their interfaces. A central database manages all data needed for the unit test.



The systematic test case determination is carried out with the help of the Classification Tree Editor **CTE,** which is fully integrated into **Tessy**.

Test data and expected values are entered into the test data editor, which allows you to browse the interface of a test object down to the basic data types. The remaining testing activities, e.g. test execution, test evaluation and test documentation, are fully automated.

Classification Tree Method and Classification Tree Editor

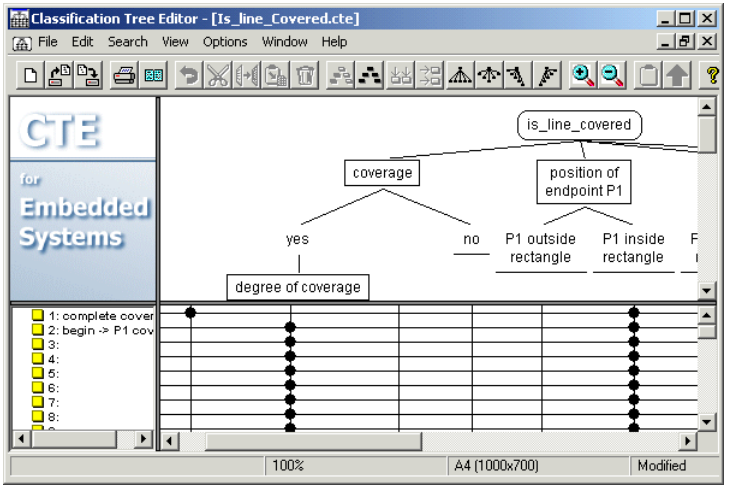
Developed in DaimlerChryslers software laboratories the **Classification Tree Method** is used for the systematic determination of test cases low in redundancy and error sensitive, in this way enabling an extensive and effective testing of a test object.

The basic concept of the Classification Tree Method is to first partition the set of possible inputs for the test object separately and from different aspects, and then to combine them to obtain redundancy-free test cases covering the complete input domain.

First of all specify the relevant aspects for the test. Each aspect must be clearly defined to allow for a clear differentiation of all possible inputs for the test object. Then the partition of the set of all possible inputs for every aspect is carried out. This partition is a **Classification** in the mathematical sense, i.e. the set of possible inputs is partitioned disjoint and completely into subsets, the so-called **Classes**. The partition is fairly easy to carry out since it is done for each aspect separately. Each aspect will then have a classification.

Combining classes of different classifications where exactly one class from every established classification is considered creates a **Test Case**. In other words, a test case is the intersection of the selected classes. It is necessary to heed the logical compatibility, i.e. the intersection may not be empty. Select as many combinations as test cases in order for all aspects to be considered sufficiently.

The **Classification Tree Editor CTE** has been developed for the efficient application of this method. It is a syntax-controlled, graphical tool that supports test case determination with the Classification Tree Method. The CTE supports you during both phases, i.e. during the classification tree design and when defining test cases in the combination table. The editor offers two work areas for these tasks where the tree and the table can be edited interactively.



Automated Regression Testing

Defining a systematic test is very time consuming and thus only justifiable if test cases and test data can be reused for a new test.

In order to appropriately react to changes of the interface of a function the test system needs to know the structure of the interface. Only by knowing the structure of the old interface is it possible to test whether the interface of a function has changed in a new version of the software. **Tessy** examines the source files and determines the functions and their interfaces by analyzing the program codes. This information is stored in a special database and can be retrieved any time.

This information enables the representation and adjustment of the interface structure independent of the test data or expected values. All settings on the interface page (e.g. the definition of external variables or stub functions) can be specified separate from the data required for the test.

The test data itself is entered in a special editor, the test data editor TDE. The data is assigned to the respective interface elements via internal references. This separation of the interface information and its test data achieves a clear distinction between structure and data.

This offers an enormous advantage when handling changes in the source codes. On the one hand the test of the interface is possible, indicating changes in the first place. On the other hand, if a change occurs usually only a few elements of the interface of a function under test are changed.

Consequently the procedure when handling interface changes is simple: With a special tool (the Interface Data Assigner, IDA) the elements of the new interface can be assigned to the elements of the old one. This assignment can be carried out automatically; new elements are simply assigned by using Drag-and-Drop.

History

The test system **Tessy** was developed by the Research and Technology Group of DaimlerChrysler.

The developers of the method and tool at DaimlerChrysler are:

Klaus Grimm
Matthias Grochtmann
Roman Pitschinetz
Joachim Wegener

Tessy has been well-tried in practice at DaimlerChrysler and is since applied successfully.

Tessy is commercially available since Spring 2000 and is further developed and marketed by ATS Software. Since 2001 the company trades under the name Razorcat Development GmbH.

New Features in Version 2.6

2008 :: DocRev. 1.1.4This Version of **Tessy** has some significant improvements:

- Enhanced Coverage Visualization and Reporting
- Automatic Test Data Generation
- Graphical Reporting of Variable Values
- C++ Support
- Using Symbolic Constants as Test Data
- Bitcheck
- Value for the Return Value of Functions in the CTE
- Interface Database Optimization
- Tool Qualification Package
- Support For Windows Vista
- New Compiler and Debugger Integrations

Important Note: *The new formats of the databases used by Tessy are not compatible with earlier versions. Tessy will prompt you to convert a project database to the new format in case you want to open an existing project database with an older format. After this conversion, it´s no longer possible to open the new database using an earlier version of Tessy.*

Enhanced Coverage Visualization and Reporting

In addition to the existing coverage reports, Tessy provides graphical visualization and reporting for coverage measures in form of a flowchart.

The branches of the C1 coverage are highlighted in red and green indicating their coverage result. Decisions of the program are also highlighted in red and green according to their coverage result for MC/DC and MCC coverage. The coverage tables for MC/DC and MCC are also available for each decision of the test object.

:: View Coverage Results

Automatic Test Data Generation

A context menu in the Test Data Editor (TDE) allows automatic generation of test data. The test data manifests in the test steps of a single test case. The test steps are generated by combination of values for the variables in the interface of the test object.

The user may simply specify to use minimum and maximum values; alternatively, the user may specify a list of values to be used for the combinations. Tessy calculates the number of test steps resulting from the current settings. This allows the user to avoid inflation of test steps by adjusting the settings. Eventually, the user may append the generated test steps to existing test steps of this test case or the user may create a test case consisting purely of generated test data.

:: Automatic Test Data Generation

Graphical Reporting of Variable Values

The test data editor TDE allows graphically visualizing test data for interface variables as signal data row. This is available for data rows of a variable over all test steps for each test case or as a whole for all test cases. The graphical result plots of interface variables are available without additional software (e.g. Matlab).

:: Test Reports with Result Plots

C++ Support

Tessy allows testing of C++ modules for certain compilers supporting the C++ language. Test data entry and result evaluation are accomplished in the Usercode of Tessy.

:: Please refer to the application notes ”Using C++”.

Using Symbolic Constants as Test Data

Tessy is able to use #defines of constant values as symbolic test data. E.g., if the user’s source code contains `#define RED 1, #define GREEN 2` and `#define BLUE 3` up to now, “2” has to be entered in Tessy’s Test Data Editor to denote “`GREEN`”.

In Tessy V2.6, “`GREEN`” can be entered directly as test data. This eases the definition of the test cases and helps to understand their semantics. Furthermore, if the definition of “`GREEN`” should change during development, e.g. to `#define GREEN 20` the value for `GREEN` in the test data will change automatically, too.

:: Using Symbolic Constants

Bitcheck

Individual bits of a variable can be checked during the evaluation for a test case result. It is now possible to mask-out bits of an expected value in the Test Data Editor. Fore this, a new format for expected values had been introduced, e.g.

```
0b000xxxx1111000
```

where "`0b`" is the format specifier, "`0`" denotes an unset bit, "1" denotes a set bit, and "`x`" denotes a don't care bit in the expected result.

:: Bitcheck

Value for the Return Value of Functions in the CTE

If a function is not void, but has a return value, it is now possible to associate a classification with the return value and to specify a value for the expected result in the underlying classes. This value will be automatically used in the test cases exported to Tessy. This is similar to what is already available for normal output variables.

Interface Database Optimization

The improved Interface Database (IDB) now allows restoring the original passing directions after any changes to the interface settings.

:: Reset Passing Direction to the Initial Value

Also new interface analysis findings of Tessy due to source code changes will be reflected within the original passing directions settings after re-use operations.

Variables and functions used or referenced within initializing statements will be available within the IDB. Optimization of the IDB database will provide faster response times for larger interfaces.

Tool Qualification Package

A tool qualification package for use within certification activities according to standards like DO-178B, IEC 61508 or Automotive SPICE is now available. The package includes a well documented suite of tests for checking the certification relevant input/output/evaluation/reporting functionality of Tessy. Customer specific adaptations or consulting services may also be offered.

:: For more information, please contact support@razorcat.com.

Support For Windows Vista

Tessy will be able to run under Windows Vista. Please refer to chapter Pre-Installation Checklist.

New Compiler and Debugger Integrations

Check for new integrations on the Razorcat home page at http://www.razorcat.com.

Conventions

Select **Tools|Batch Test…** indicates a menu selection followed by a submenu selection. In this case, select **Tools** from the menu bar, then select **Batch Test…** from the drop-down menu.

NOTE: *Indicates important or supplemental information.*

TIP: *Indicates suggestions that may be helpful.*

`Fixed-Space Font` represents information that you need to type and messages from the system. The fixed-space font is Courier New.

The manual instructions assume that you installed Tessy in the default location. If you installed in another location, you must navigate to files in that location instead of the location specified in the instructions.

Installation

Preparing for the Installation

The Tessy version 2.6 and later allows you to have multiple Tessy installations with different versions on the same computer. Depending on the actual installation situation, there are different dialogs and settings required during setup (“Installing Tessy” on page 14).

Note: *You need local administrator privileges on the system in question to perform the installation of Tessy.*

Pre-Installation Checklist

System Requirements

- Tessy requires Windows 2K (SP4 and higher), Windows XP (SP2 and higher) or Windows Vista.

  The Floating License Server may also be installed on Windows Server 2000 or 2003 (s. Install Central License Server).

- Automatic aliases should be enabled (recommended).

```
Hive: HKEY_LOCAL_MACHINE
Key: SYSTEM\CurrentControlSet\Control\FileSystem\
Name: NtfsDisable8dot3NameCreation
Type: REG_WORD
Value: 1 Disable automatic alias
Value: 0 Enable automatic alias
```

  Setup will check this and will enable automatic aliases if necessary. You have to reboot your machine.

Note: *Tessy converts all path names to 8.3 short names by using a function provided by Windows. This conversion could fail if automatic aliases have been disabled. Normally, this option is enabled by default (value = 0).*

- We recommend at least a 1.5 GHz CPU and 1 GB RAM for Tessy. This is dependent of the Operation System used.

Software Requirements

- Tessy requires MS Word 2000/2002/2003 and MS Excel 2000/2002/2003 to be installed for report generation, if you want to use these report formats (s. Test Reports).
- You may also use a web-browser, if you choose HTML as report format. Tessy supports standard web-browsers, e.g. MS Internet Explorer, Firefox. Your browser must have both cookies and JavaScript enabled.
- If you want to use CHM (compiled HTML) as your report format, it must be installed HTML Help Workshop. This tool can be downloaded under www.Microsoft.com.
- Tessy requires Python version 2.2.1, Python Win32 extensions version 1.48 and at least Java Runtime Version SE 1.5.0_04 and Cygwin version 1.5.18 to be installed (see Using Cygwin Versions other than Cygwin 1.5.18). Setup will install Cygwin

Version 1.5.25 if necessary. You do not need to update an already installed Cygwin version 1.5.18 (from former Tessy installations) on Cygwin version 1.5.25.

- If you use Tessy with Microsoft Vista, you must install at least Cygwin Version 1.5.25. Setup will install this version if necessary.

***Note**: Please mind, that other versions of Python and Cygwin could cause problems during report generation respectively test execution!*

Setup installs not an entire Cygwin and Python development environment, including only the necessary packages for use with Tessy. You may use your already installed versions, but this versions must be at least 1.5.18 (see Using Cygwin Versions other than Cygwin 1.5.18) respectively 2.2.1. You may install Python 2.2.1 parallel to your used version.

- Tessy requires the installation of compilers, debuggers and emulators which are intended to be used with Tessy. After the installation of Tessy you have to configure Tessy by means of the **Tessy Environment Editor** to use these tools.

  Please check for updates of supported environments at www.razorcat.com.

***Note**: The configuration of an earlier version of Tessy is taken over by the installation of Tessy 2.6 (s. Update a Previous Installed Version).*

Firewall and Virus Scanner

We strongly recommend to deactivate a running firewall or virus scanner temporarily on your PC. This may prevent problems during the installation of Tessy.

Note that some anti-virus software and firewalls can limit the functionality of a range of software applications, including Tessy.

If your anti-virus software or firewall alerts you with a warning while using Tessy, it may be necessary to modify the properties of your anti-virus software or firewall.

Performing Silent Installations

You may install Tessy silently with a command line. For more information, please refer to the application notes Installing Tessy from **Help|Documents**. Setup includes also a msi package which has been generated by InstallAware.

Installing Tessy

Note: *You need local administrator privileges on the system in question to perform the installation of Tessy.*

When installing Tessy 2.6, there may be one of the following situations on the targeted computer:

- No Tessy installation present (New Installation on page 15)
- An older Tessy installation (e.g. version 2.5.x, 2.6.1, etc.) present (Update a Previous Installed Version on page 31)
- Tessy version 2.6. present (Multiple Installations of Tessy 2.6.x on page 32)

New installation and upgrade from a previous version of Tessy (e.g. 2.6.17 > 2.6.18) are treated the same way. On an update installation, some of the installation steps described in the following chapters will be omitted.

Setup Overview

Tessy setup will install the following components by default:

- **Tessy** files
- **CTE** files
- **Floating license server** (FLS) files

You may choose to install the FLS files (for license server setup) or the CTE files only. A complete Tessy installation requires all components to be installed (s. Choose Setup Type on page 22).

The files will be installed into the following directories:

- A **base** directory, e.g. `C:\Program Files\Razorcat`
- A **version** directory.
  Setup will always use `Tessy_2.6` as the default version directory (`[base directory]\Tessy_2.6`).

The **base** directory will be chosen during the initial setup of Tessy and is modifiable by the user. The version directory is a presetting of Setup and cannot be changed. The name of the directory depends on the used major version of Tessy, e.g. `Tessy_2.6`. (s. Multiple Installations of Tessy 2.6.x on page 32).

The above mentioned components will be installed as follows:

- **Tessy** files into the **version** directory
- **CTE** files into the **version** directory
- **FLS** files into the **base** directory

If you want to add or remove components to an existing installation, you also need to start the setup program. Please refer to section "Modify, Repair or Remove Installation" on page 34.

Important Note: *The new formats of the databases used by Tessy v2.6 are not compatible with earlier versions. Tessy will prompt you to convert a project database to the new format in case you want to open an existing project database with an older format. After this conversion, it´s no longer possible to open the new database using an earlier version of Tessy.*

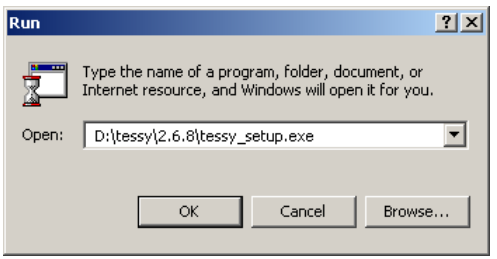
New Installation

To install **Tessy** on your computer, please insert the **Tessy** CD into the CD-ROM drive and wait for the setup program to start.

- Select from the options provided to start the Tessy installation.

  If setup will not start automatically, please press the **[Start]+[R]** key and either choose **Browse...** or type into the command line

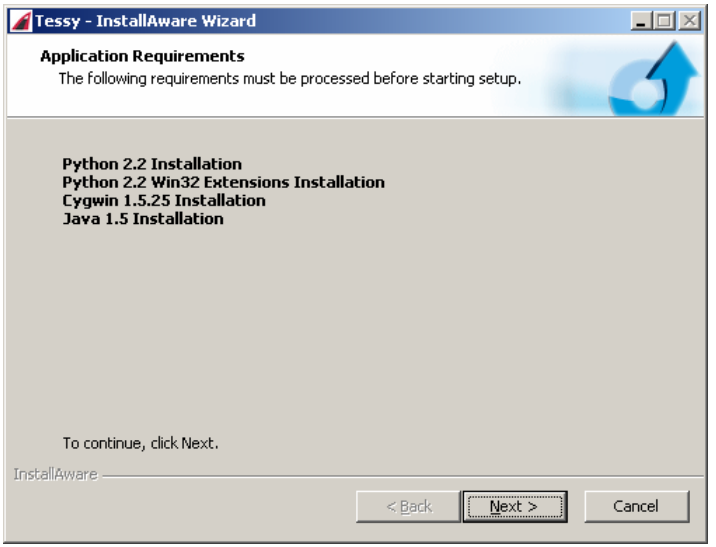
  ```
  [CD Drive]:\Tessy\[TessyVersion]\tessy_setup.exe
  ```



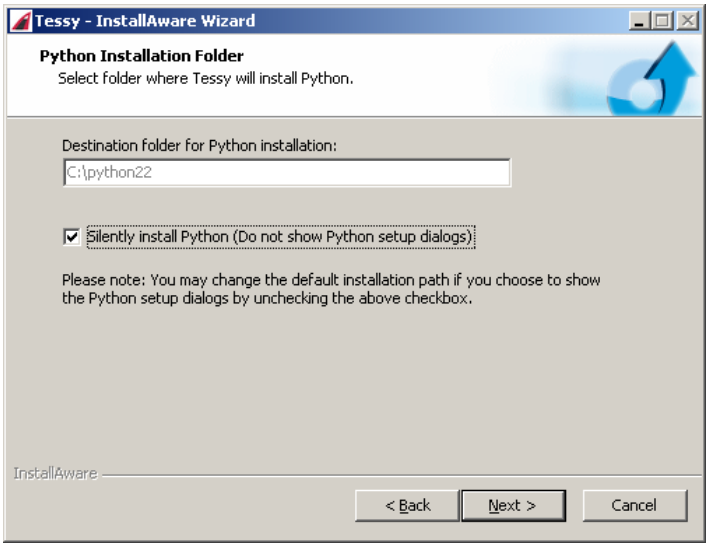
- Click on **OK** and setup will be started.

Installing Tessy Requirements

- Python, Cygwin and Java must be installed for Tessy. Setup will install all applications if necessary. If already installed some or all of the installation steps will be omitted. Setup will continue then with step "Installing Tessy" (s. below).



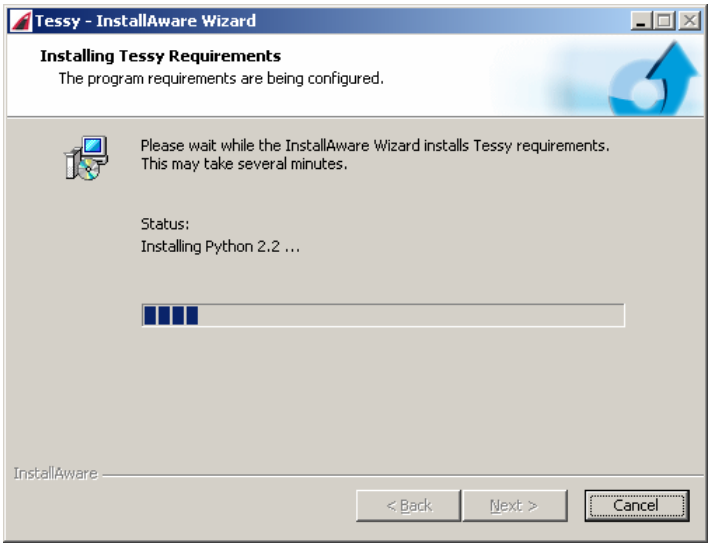
- Click on **Next** to continue.
- **Installing Python**


Tessy - InstallAware Wizard
Python Installation Folder
Select folder where Tessy will install Python.
Destination folder for Python installation:
C:\python22
Silently install Python (Do not show Python setup dialogs)
Please note: You may change the default installation path if you choose to show the Python setup dialogs by unchecking the above checkbox.
InstallAware
< Back
Next >
Cancel

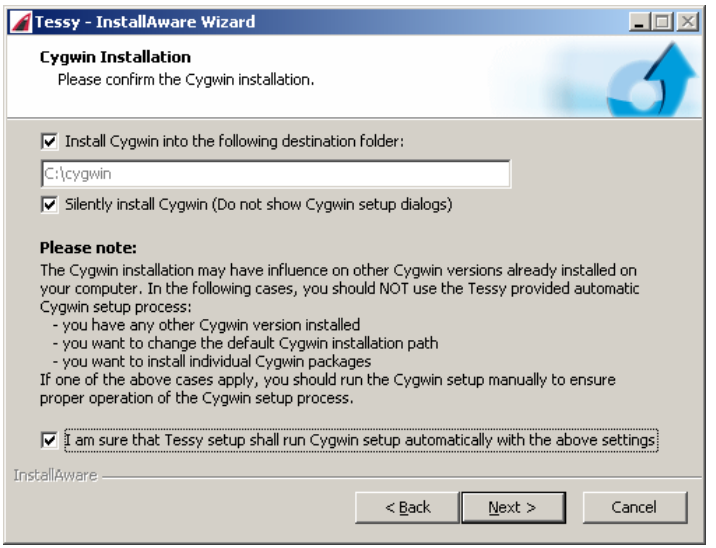
If you want to choose another destination folder, please deselect “Silently install Python”. Follow the instructions on the screen to complete the Python installation.

***Note**: Other versions of Python could cause problems! You may install Python 2.2.1 parallel to your installed version, e.g. 2.4.*

- Click on **Next** to continue.



- **Installing Cygwin**

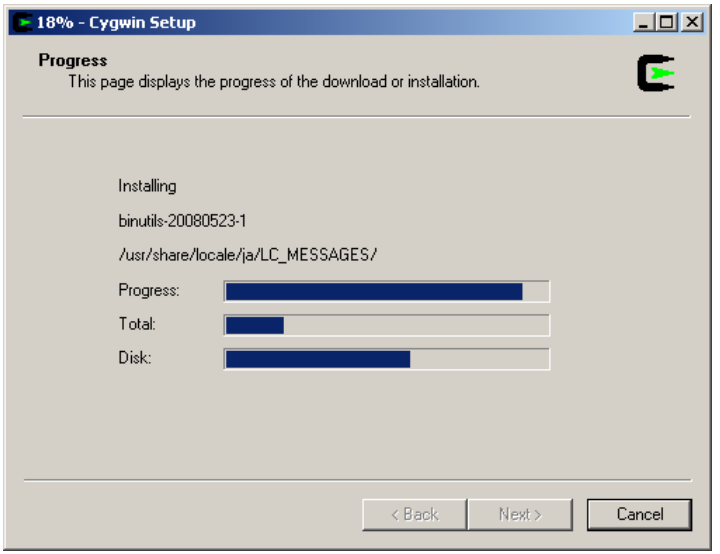

Tessy - InstallAware Wizard
Cygwin Installation
Please confirm the Cygwin installation.
Install Cygwin into the following destination folder:
C:\cygwin
Silently install Cygwin (Do not show Cygwin setup dialogs)
Please note:
The Cygwin installation may have influence on other Cygwin versions already installed on your computer. In the following cases, you should NOT use the Tessy provided automatic Cygwin setup process:
- you have any other Cygwin version installed
- you want to change the default Cygwin installation path
- you want to install individual Cygwin packages
If one of the above cases apply, you should run the Cygwin setup manually to ensure proper operation of the Cygwin setup process.
I am sure that Tessy setup shall run Cygwin setup automatically with the above settings
InstallAware
< Back
Next >
Cancel

If you want to choose another destination folder, please deselect “Silently install

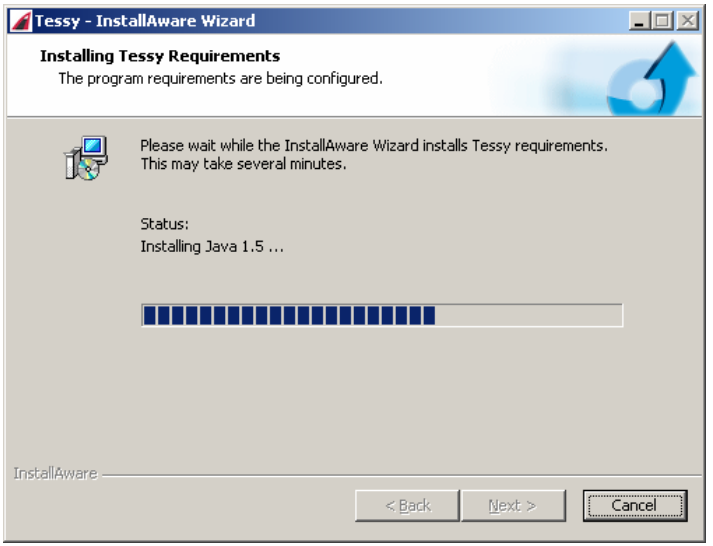
Cygwin". Follow the instructions on the screen to complete the Cygwin installation.

***Note**: Please mind: Setup installs not an entire Cygwin development environment, only the necessary packages for use with Tessy. You may use your already installed version, but this version must be at least 1.5.18. Other versions of Cygwin could cause problems during test execution (s. Using Cygwin Versions other than Cygwin 1.5.18)! You do not need to update an already installed Cygwin version 1.5.18 (from former Tessy installations) on Cygwin version 1.5.25.*

- Click on **Next** to continue.



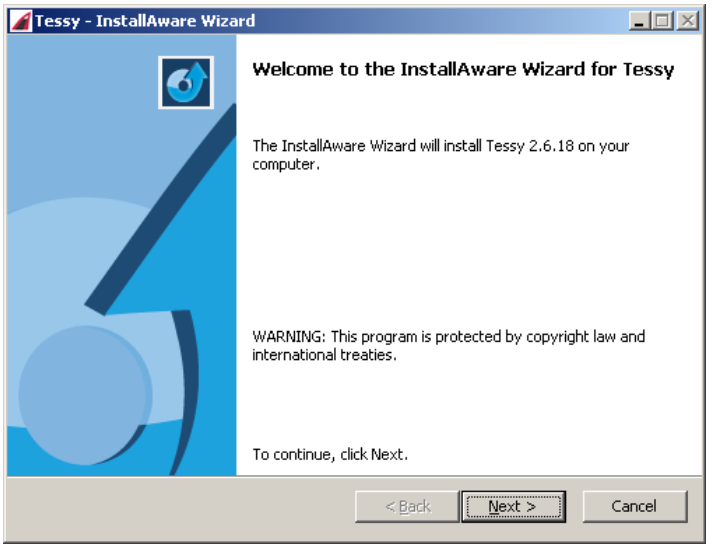
- **Installing Java**



As soon as the installation of Tessy Requirements is finished, Setup will continue with the installation of Tessy (s. Installing Tessy).

Installing Tessy

- The following steps will guide you to install Tessy.


Tessy - InstallAware Wizard
Welcome to the InstallAware Wizard for Tessy
The InstallAware Wizard will install Tessy 2.6.18 on your computer.
WARNING: This program is protected by copyright law and international treaties.
To continue, click Next.
< Back
Next >
Cancel

- Click on **Next** to continue.

License Agreement

Please read the license agreement carefully. To install Tessy, you must accept this agreement.

Tessy - InstallAware Wizard
License Agreement
Please carefully read the following license agreement.
Tessy V2.6
(c) 1998-2008, Razorcat Development GmbH and Daimler AG
License Agreement
==========
Razorcat Development GmbH (in the following named Razorcat), provides the guarantee, and the licensee, according to the following terms, accepts a non-exclusive licence or sublicence to utilize and reproduce software in binary format, developed by Razorcat or a third party and distributed by Razorcat or a third party.
Conditions of Use
==========
I accept the terms of the license agreement
InstallAware
< Back
Next >
Cancel

- Click on **Next** to continue.

Choose Setup Type

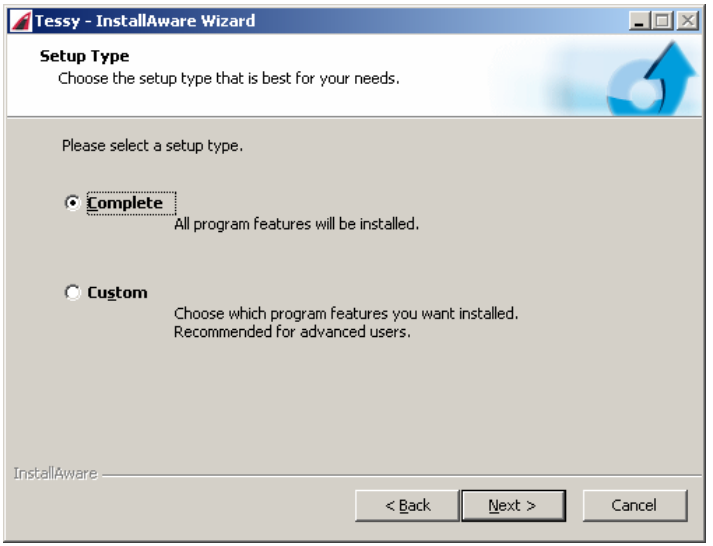
Complete

- You may choose **Complete** to install all required components of Tessy (recommended). A complete installation includes:

  Tessy files
  These are the main components of Tessy.

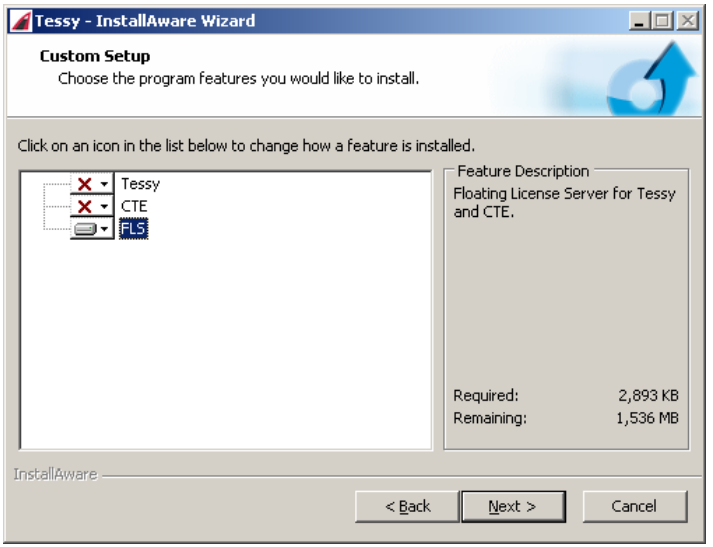
  CTE files
  These are necessary files for the Classification Tree Editor (CTE). With CTE you determine test items and import them to Tessy.

  Floating License Server
  The Floating License Server is necessary to use a local license with Tessy. FLS will be installed always automatically. The FLS files will be installed into the base directory (s. Setup Overview on page 14).



Custom

- You may choose **Custom** to install only selected components, e.g. **CTE** or Floating License Server **FLS**.



Note: *You may also use the standalone setup for the FLS if you want to install the FLS on a central server in your network. This setup package can be downloaded from www.razorcat.com. (s. also "Install Central License Server" on page 43)*

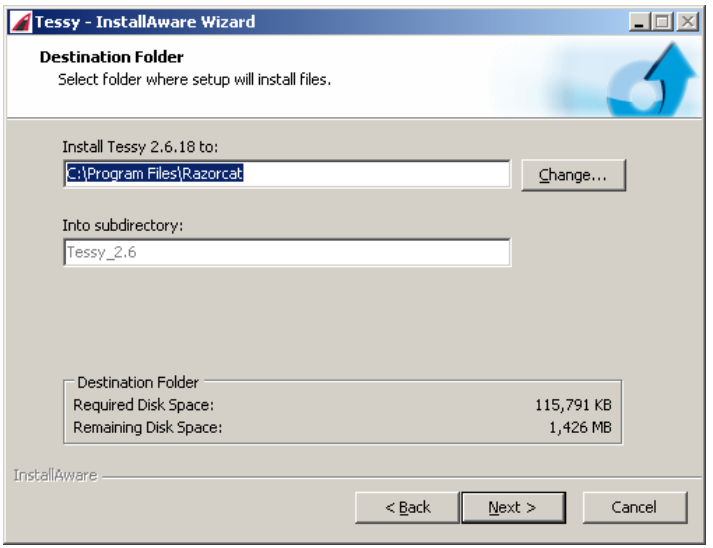
Choose Destination Location for Tessy

- Within the destination folder dialog, you may select the base directory where all the files will be installed. The default base directory is:

  `C:\Program Files\Razorcat`

  Setup will install the Tessy and CTE files in a version directory named t**essy_2.6** by default. This version directory cannot be selected.

  `C:\Program Files\Razorcat\tessy_2.6`

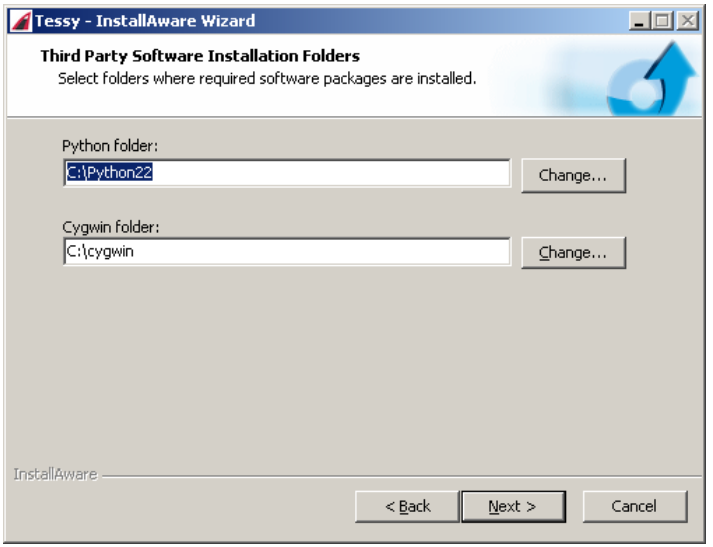


- If you choose the default folder, click on **Next** to continue.

Choose Third Party Installation Folders

You may select your preferred Python or Cygwin installation folders. Normally Setup has both applications already installed if necessary. Please accept then the default paths and click next to continue.

Note: *Tessy requires Python Version 2.2.1, Python Win32 extensions Version 1.48 and at least Cygwin version 1.5.18 (2K, XP) or version 1.5.25 (Vista) to be installed.*

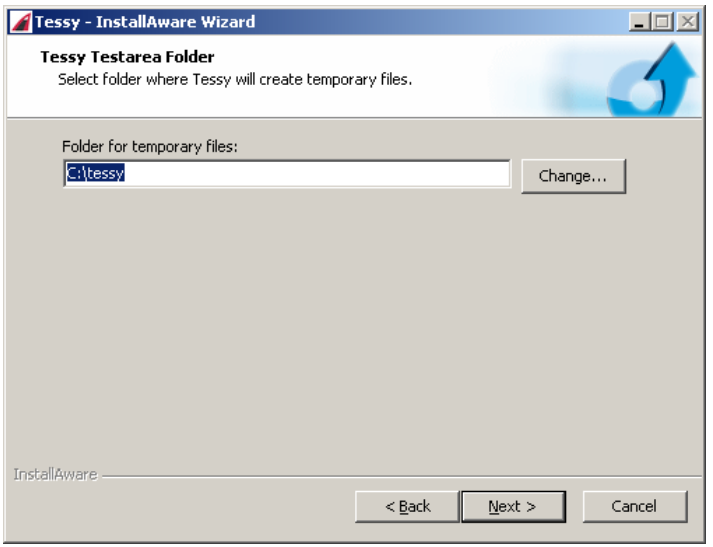


- Click on **Next** to continue.

Select Testarea

- In the next dialog you will be prompted for the *Tessy Testarea*, a dedicated directory, where Tessy will create **temporary** files for test execution. Please choose a path **without blanks** or accept the default `C:\tessy\.` Setup will then create several subdirectories like `C:\tessy\testarea.`

Note: *Please choose another path if the user owns only limited rights on this disk drive!*

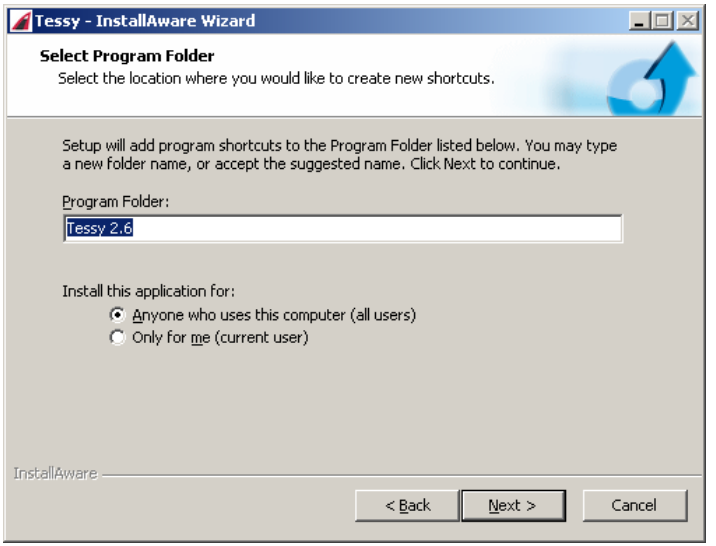


Caution: *Don't copy or store any files to* `$(TESSY_TESTAREA)\testarea` *(by default:* `C:\tessy\testarea)`*. The whole content of this folder will be deleted by Tessy before test execution!*

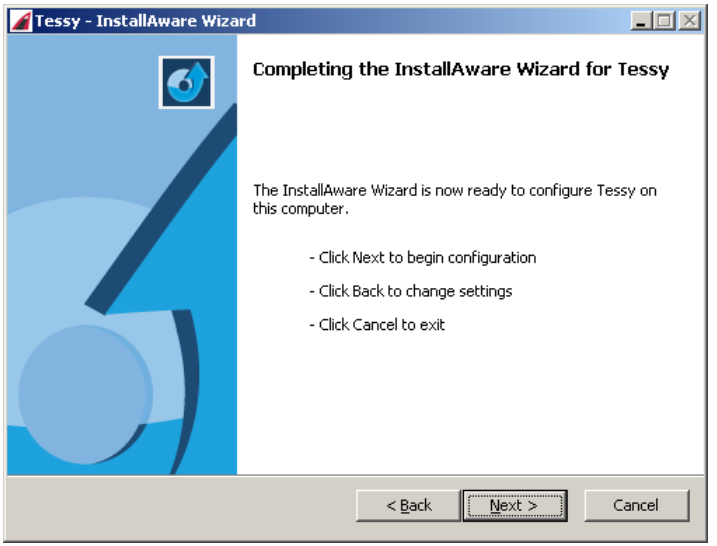
- Click on **Next** to continue.

Select Program Folder

- In this dialog you specify the Folder within the *Windows Start Menu* where all Tessy shortcuts should be created. We suggest accepting the default folder **Tessy 2.6**.


Tessy - InstallAware Wizard
Select Program Folder
Select the location where you would like to create new shortcuts.
Setup will add program shortcuts to the Program Folder listed below. You may type a new folder name, or accept the suggested name. Click Next to continue.
Program Folder:
Tessy 2.6
Install this application for:
Anyone who uses this computer (all users)
Only for me (current user)
InstallAware
< Back
Next >
Cancel

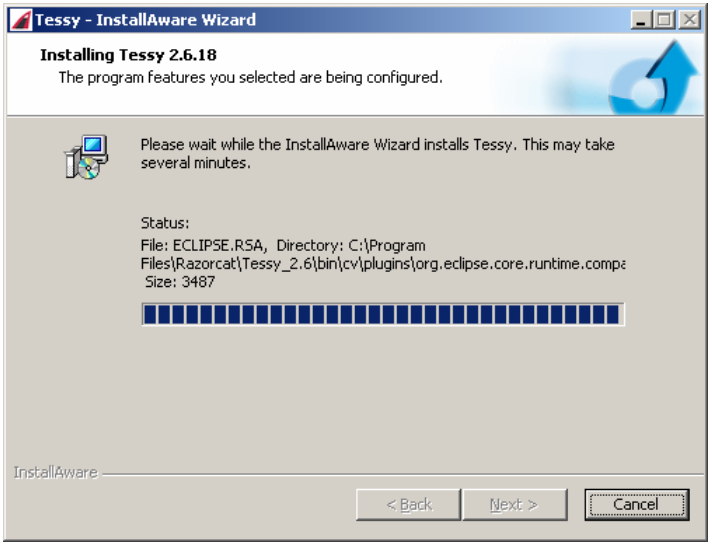
- Click on **Next** to continue.


Tessy - InstallAware Wizard
Completing the InstallAware Wizard for Tessy
The InstallAware Wizard is now ready to configure Tessy on this computer.
- Click Next to begin configuration
- Click Back to change settings
- Click Cancel to exit
< Back
Next >
Cancel

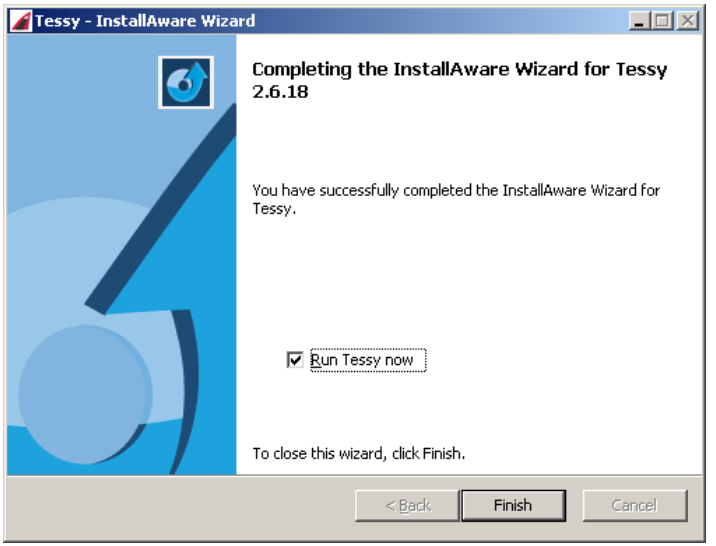
- Click on **Next** to begin the configuration.

The Installation Process

- Now **Setup** copies all files to your disk into the specified installation path.


Tessy - InstallAware Wizard
Installing Tessy 2.6.18
The program features you selected are being configured.
Please wait while the InstallAware Wizard installs Tessy. This may take several minutes.
Status:
File: ECLIPSE.RSA, Directory: C:\Program Files\Razorcat\Tessy_2.6\bin\cv\plugins\org.eclipse.core.runtime.compa
Size: 3487
InstallAware
< Back
Next >
Cancel

- On completion of the installation process, the following dialog will appear:


Tessy - InstallAware Wizard
Completing the InstallAware Wizard for Tessy 2.6.18
You have successfully completed the InstallAware Wizard for Tessy.
Run Tessy now
To close this wizard, click Finish.
< Back
Finish
Cancel

- Click on **Finish**. Tessy is now installed on your computer.

Note: *Next, you have to register Tessy to get it running (see Registration on page 36).*

Using Cygwin Versions other than Cygwin 1.5.18

Tessy requires at least Cygwin Version 1.5.18 to be installed. The Tessy Setup will install Cygwin version 1.5.25 if necessary, including only the necessary packages for use with Tessy. Unfortunately, other versions of Cygwin could cause problems during test execution.

Versions less than 1.5.18

If you need to have a Cygwin Version less than 1.5.18 installed, please refer to the application notes GNU PPC Compiler for more information on this topic. You need to install Cygwin 1.5.18 in parallel to your preferred Cygwin version.

Versions greater than 1.5.18

You may use your preferred Cygwin version with Tessy but it must be at least version 1.5.18 or greater.

Normally, Setup will find an already installed version of Cygwin. If necessary, change the path in the Third Party Installation Folders dialog to fit your needs (s. Choose Third Party Installation Folders on page 25).

***Note**: Please make sure that you have installed the necessary Cygwin packages for use with Tessy. For this purpose, please refer to the application notes Installing Tessy under **Help|Documents**.*

Edit tessy.conf

Tessy needs a special setting to use Cygwin versions greater than version 1.5.18. Normally this setting has been enabled by default. If you encounter problems, please check the following:

- Open the file `tessy.conf` from Tessy by using **Options|Edit Settings...** (Setup has copied `tessy.conf` automatically from ..**\Razorcat\Tessy_2.6\config** into **%APPDATA%\Razorcat\Tessy\ [Version]\config** when you open Tessy the first time.)
- Navigate to the section `[Environment Variables]`.
- Remove the # in front of `TESSY_DONT_CHECK_CYGWIN_VERSION=1` if still commented out.
- Save the file.

Update a Previous Installed Version

The installation process will be nearly the same as for a new installation described on page 15 New Installation. Some of the installation steps will be not executed.

Setup will uninstall the previous version of Tessy 2.6 before installing the new version.

The target environment configuration of an installed earlier version of Tessy is taken over by the installation of Tessy 2.6 automatically. If you have changed default files in earlier versions of Tessy (e.g. make-templates, linker files, etc) you have to carry out these changes in Version 2.6 manually.

Tessy provides the so called Tessy Environment Editor **TEE** to configure supported target environments and compilers (s. TEE).

Setup will always recommend the base directory of the first installed version of Tessy, e.g. "`C:\Program Files\Razorcat`". We recommend choosing the given base directory. This will avoid unexpected results using the Modify or Repair option of Setup, if necessary.

Important Note: ***The new formats of the databases used by Tessy v2.6 are not compatible with earlier versions. Tessy will prompt you to convert a project database to the new format in case you want to open an existing project database with an older format. After this conversion, it´s no longer possible to open the new database using an earlier version of Tessy.***

Update a Previous Installed Version of Tessy less than 2.6

The installation process will be nearly the same as for a new installation described on page 15 in chapter New Installation. Some of the installation steps will be not executed.

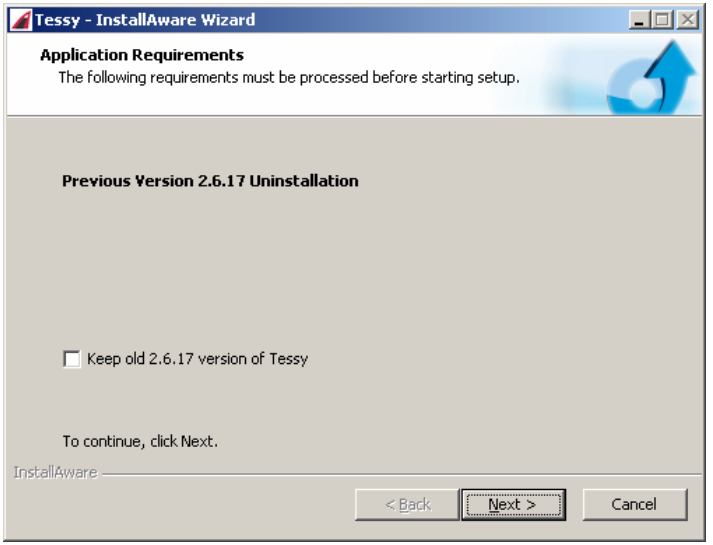
An update will preserve your previous installed Tessy version (e.g. 2.5) and will choose another installation folder for the new version 2.6 of Tessy. You may also use the previous version for testing. Please refer also to chapter "Multiple Installations of Tessy 2.6.x on page 32".

Update and Replace a Previous Installed Version of Tessy 2.6

The installation process will be nearly the same as for a new installation described on page 15 in chapter New Installation. Some of the installation steps will be not executed.

- Start Setup as described on page 15 "New Installation" and follow the instructions.

- In the Application Requirements dialog, click next to update a previous installed version of Tessy 2.6. The check-box *"Keep old version of Tessy"* remains unchecked.



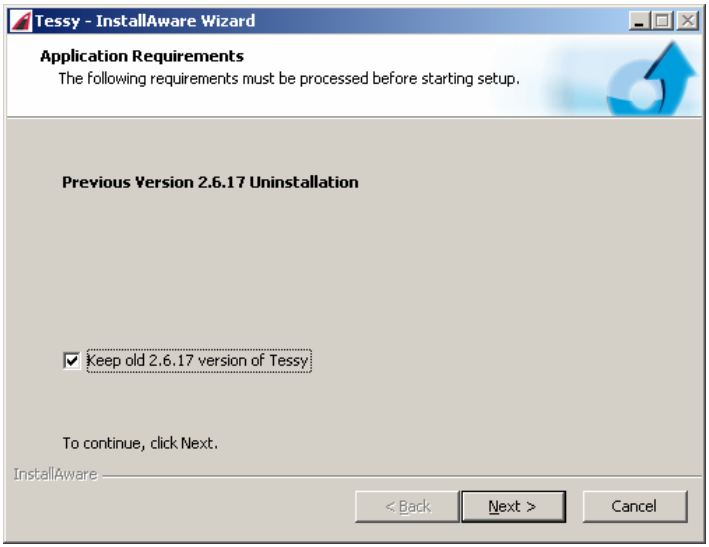
If you want to keep the old version of Tessy 2.6, please continue with Multiple Installations of Tessy 2.6.x.

***Note**: If you have changed default files in earlier versions of Tessy (e.g. make-templates, linker files, etc), please save these templates before you update your Tessy installation. Setup will uninstall the previous version of Tessy 2.6 before installing the new version!*

Multiple Installations of Tessy 2.6.x

The installation process will be nearly the same as for a new installation as described in chapter New Installation on page 15. Some of the installation steps will be not executed.

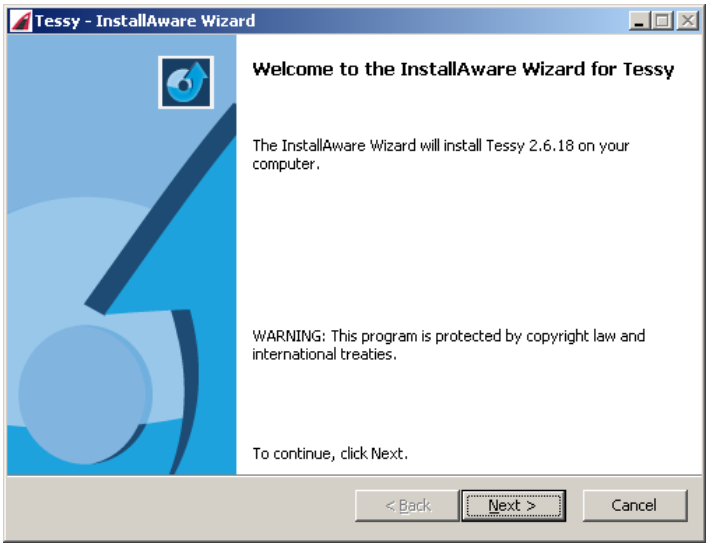
- Start Setup as described on page 15 "New Installation" and follow the instructions.
- In the Application Requirements dialog, tick the check-box *"Keep old version of Tessy"*.



- Click next to update a previous installed version of Tessy 2.6, e.g. **2.6.17.**

  Setup will rename the version directory **.\Razorcat\Tessy_2.6** in **.\Razorcat\Tessy_2.6.17** to keep the old version **2.6.17** of Tessy.

  Setup will create a new shortcut in the **Program** folder of the Windows **Start** menu for the previous installed version, e.g. **Tessy 2.5.17**


Tessy - InstallAware Wizard
Welcome to the InstallAware Wizard for Tessy
The InstallAware Wizard will install Tessy 2.6.18 on your computer.
WARNING: This program is protected by copyright law and international treaties.
To continue, click Next.
< Back
Next >
Cancel

- In the Welcome dialog, click **Next** to continue the installation of the new version of Tessy.
- Follow the steps as described under "Installing Tessy" on page 14.

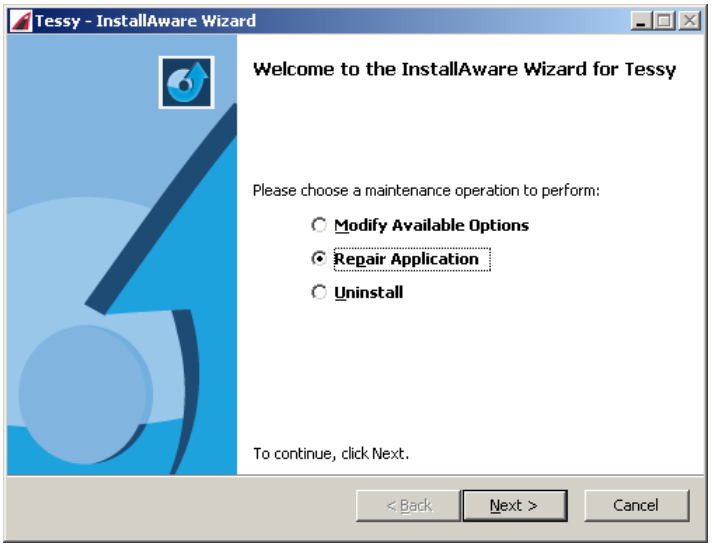
***Note**: Setup will always use the base directory of the first installed version of Tessy, e.g. "`C:\Program Files\Razorcat`". This base directory is not changeable!*

The target environment configuration of an installed earlier version 2.6.x of Tessy is taken over by the installation of the new version of Tessy automatically if you start Tessy the first time.

If you have changed default files in earlier versions of Tessy (e.g. make-templates, linker files, etc) you have to carry out these changes in Version 2.6 manually. Tessy provides the so called Tessy Environment Editor **TEE** to configure supported target environments and compilers (s. TEE).

Modify, Repair or Remove Installation

If you run Setup once again and Tessy has been already installed you will see a dialog with the following options.



- **Modify:** Add or remove components of the **last-installed** Tessy version (s.Multiple Installations of Tessy 2.6.x on page 32).

  Repair: Reinstall the **last-installed** version of Tessy with the same configuration.

  Uninstall: Removes the **last-installed** version of Tessy.
- Select one of the options and click on **Next**.

Registration

The licensing for the products CTE und Tessy uses the *Floating License Server* developed by Razorcat.

Depends on your purchased license model there are two types of licenses possible:

- Floating License(s)
- Node-Locked License

The floating license is a server license issued for a given host id for a dedicated server within your network. It's possible to share the license(s) with other users within your network.

The node-locked license is a single user license issued for a given host id. It's not possible to share the license with other users within your network. A node-locked license operates only on the particular machine for which the license is issued.

Floating License

The *Floating License Server* allows you to manage all licenses on a central network server. Tessy and CTE can then be used on any computer on the network. The number of users who can use the software simultaneously is determined by how many licenses you have purchased. A license manager running on the license server keeps track of the number of users.

You may also check out licenses for computers that don't have access to your network, e.g. notebooks while traveling (see Checkout Licenses on page 49).

- Please read section "Specify Central License Server" on page 42, if you want to use a floating license from an already installed license server within your company.
- Please read section "Install Central License Server" on page 43, if you want to install a central license server in your network.
- If you need a license key, refer to section "Request License" on page 37.

Node-Locked License

A node locked license can be installed on a local license server in conjunction with Tessy and CTE. A node-locked license operates only on the particular machine for which the license is issued.

- Please read section "Install Local License Server" on page 40, if you want to install a local license server on your computer.
- If you need a license key, refer to section "Request License" on page 37.

Request License

The license key for Tessy and CTE is issued for one specific PC or network server and can only be used on that particular machine.

For floating licenses, generate the license request exactly on the machine that will be your central license server within your network. For node-locked licenses, generate the license request on each workstation that will have a node-locked license.

If you are upgrading from an earlier version of Tessy or if there is a license server currently running on your network, you may not need to request a license at this time (unless the major release number has changed, e.g. 2.5 > 2.6).

License(s) for Purchaser

To request a permanent license file from Razorcat, please provide Razorcat following information:

- **MAC** address of your PC where you intend to run either a **local** (node-locked license) or a **central** License Server (floating license).
- Invoice / Order Number
  (please see the delivery notes of your distributor)
- Contact information of a person who is responsible for Tessy/CTE (Customer Support will send licenses, updates etc. direct to this person).

Please send your request to support@razorcat.com. You will get a license key file from the Customer Support by e-mail. Please refer to section "Register Tessy" on page 40 on how to install the license key.

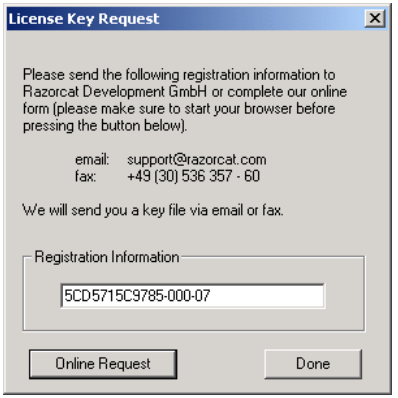
Hint: *To determine the MAC address of your PC (physical address of the network adapter), please open a command prompt window and use following command:*
`ipconfig /all`

Expiring Demo License

To request a demo license key from Razorcat, do the following:

- Start Tessy and select **Register** from the **Help** menu**.** Within the **Register** dialog click on **License...** (If you start Tessy the first time, the **Register** dialog will popup automatically) **or**
- Start the **Floating License Manager** (from folder Tessy within the Windows Start menu) and select **Request** from the **License** menu.

  You will see a dialog with the registration information.


License Key Request
Please send the following registration information to Razorcat Development GmbH or complete our online form (please make sure to start your browser before pressing the button below).
email: support@razorcat.com
fax: +49 (30) 536 357 - 60
We will send you a key file via email or fax.
Registration Information
5CD5715C9785-000-07
Online Request
Done

Please send this *Registration Information* with your contact information (your company e-mail address, company address and telephone number) either per e-mail or fax to Razorcat or click on **Online Request**. This will direct you to the Razorcat Internet registration page, where you may request an expiring demo license key.

You will get a license key file from Razorcat that you have to save to your local disk. Please refer to section "Register Tessy" on page 40 or "Install Local License Server on page 40" on how to install the license key.

Register Tessy

There are two possibilities to install the license file:

- To install a node-locked license on a local license server installation, please read section “Install Local License Server”.
- To install a server license on a central license server within your network, please read section “Install Central License Server”.

Install Local License Server

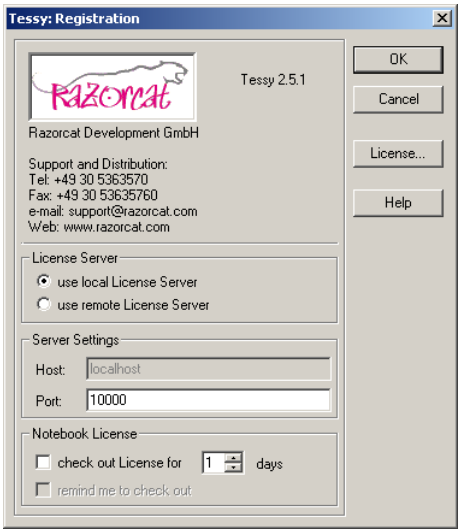
When installing Tessy, a local license server will be installed automatically. In order to run Tessy, you must obtain (see Request License on page 37) and install a valid license (see Install the License Key on page 40).

If you have a central license server within your network installed, you need only to specify the server address (see Specify Central License Server on page 42).

Install the License Key

Please take the following steps to configure a local license server:

- Save the license key file from Razorcat to your local disk, if not already done (s. Request License on page 37).
- Start Tessy either from the folder Tessy within the Windows Start Menu or use [Ctrl]+[Alt]+[T]. If you didn’t enter any license information before, you will see the following dialog:



- In section **License Server**, please select **use local License Server** (default).
- Change the **Port** number in **Server Settings**, if already in use (default: 10000).
- Click on **OK.**
  Tessy tries to start the local license server and tries to get a connection. If the license file is invalid or missing, an error message will be displayed. Click on **OK** and Tessy will display a file selection dialog, that prompts you for the license key file.
- Select your local license key file and click **OK**.

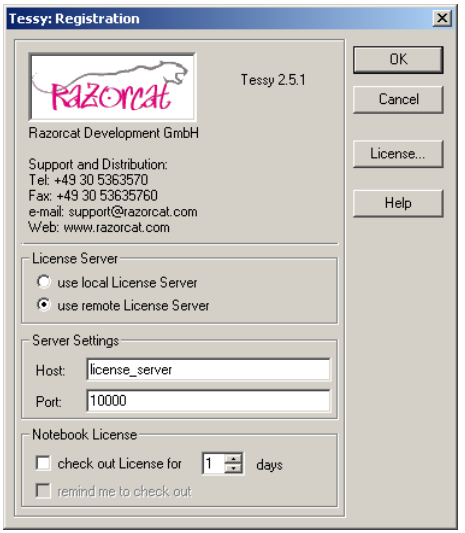
Tessy will install the license file and will try to start the license server. If an error message occurs, please see "Troubleshooting License Server on page 47".

Note: *Now, you have completed the registration and Tessy is ready to go.*

Specify Central License Server

If you have a license server running within your local network, please take the following steps:

- Start Tessy either from the folder Tessy within the Windows Start Menu or use [Ctrl]+[Alt]+[T]. If you didn't enter any license information before, you will see the following dialog:



- In section **License Server,** please select **use remote License Server**.
- In section **Server Settings,** please specify the address and port number of your central license server. Please ask your system administrator for this information.

Click on **OK.** Tessy tries to connect to the license server. If an error message occurs, please see "Troubleshooting License Server on page 47".

Note: *Now, you have completed the registration and Tessy is ready to go.*

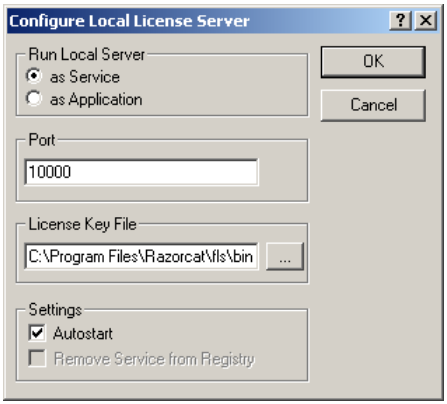
Install Central License Server

Important Note: *In order to share the license with other users within your network you need to have a server license (s. Registration on page 36).*

To install a central license server within your network, please take the following steps:

- Install the Floating License Server on your network server (refer to the chapter "Installing Tessy" on page 14).
- Start the **Floating License Manager** on that server (s. Using Floating License Manager on page 46). You will find the *License Manager* in the folder **Programs|Tessy** from the Windows Start menu.

- Click on **Configure** in the toolbar.
- The following dialog will pop up.



- Please choose following options within the dialog to run the license server as service (normally used for a network server).

Run Local Server:
As service

Port:
Change the default port number 10000, if already in use.

License Key File:

You **must** copy the license key file to the bin directory oft the license server, normally to `C:\Program files\Razorcat\fls\bin` (depending on your installation path).

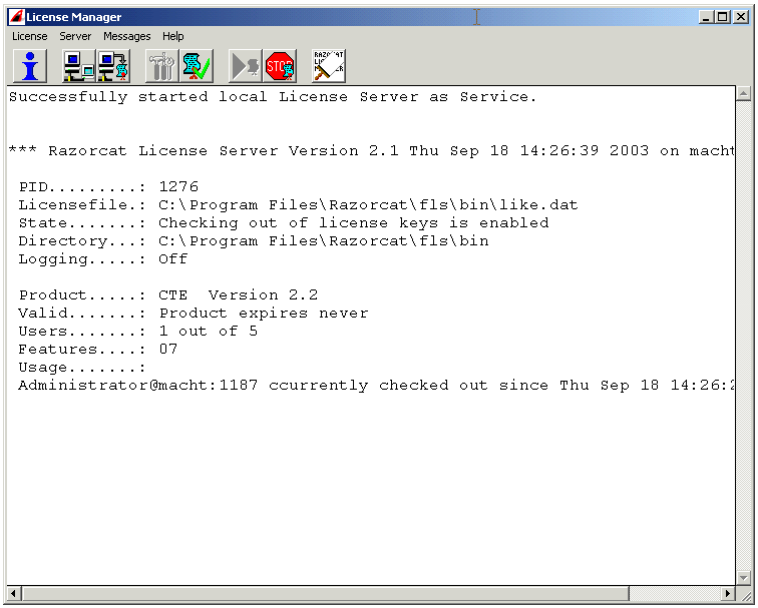
Settings:
Autostart (the license server will be started automatically during the boot process of the network server.)

- When you have finished your selection, click on **OK.** You will see <`Successfully changed configuration`> in the main window of the license manager.

- Click on **Start Local Server** in the toolbar.

You will see <`Successfully started local License Server as Service`> in the main window of the license manager.

- Click on **Info** to query the license server status.


License Manager
License Server Messages Help
Successfully started local License Server as Service.
*** Razorcat License Server Version 2.1 Thu Sep 18 14:26:39 2003 on macht
PID.....: 1276
Licensefile.: C:\Program Files\Razorcat\fls\bin\like.dat
State.....: Checking out of license keys is enabled
Directory...: C:\Program Files\Razorcat\fls\bin
Logging.....: Off
Product.....: CTE Version 2.2
Valid.....: Product expires never
Users.....: 1 out of 5
Features....: 07
Usage.....:
Administrator@macht:1187 ccurrently checked out since Thu Sep 18 14:26:

If the local license server was started, you may quit the license manager.

Using Floating License Manager

The **Floating License Manager** is used to control and to configure the **Floating License Server**.

You can start the **FLM** from the Windows Start menu: **Programs|Tessy** (default).

Normally, there is no need to use the **FLM**, because the configuration of a local license server takes place during the installation of the license file automatically (s. Install Local License Server on page 40). You may use the **FLM** to change default settings to suit your needs.

On the other hand, you need to use the **FLM** to configure a central license server (s. Install Central License Server on page 43) or in case of problems.

Configuring the License Server

Please read section "Install Central License Server" on page 43 for more information on this topic.

Starting and Stopping the License Server

To start the **Floating License Server**, do the following:

- click on **Start Local Server** in the toolbar or choose **Start Local Server** from the **Server** menu.

- click on **Stop Local Server** in the toolbar or choose **Stop Local Server** from the **Server** menu.

Checking License File

In case that the Floating License Server don't start successfully you can check the license file or view the file 'C:\Program Files\Razorcat\fls\bin\flsd.log' to see more details (s. also Troubleshooting License Server)

- To check the license file, click on **Check** in the toolbar.

Your license key from Razorcat should match one of the Host ID's listed and should be **OK.**

Example:

```
Host-ID.....: 4890A0E0B8EE : OK
Licensefile OK.
```

***Note**: You cannot transfer the license file from one to another installation of Tessy respectively license server. The license key has been issued for a specific Host ID only!*

Troubleshooting License Server

In the following you will find the most important problems which can appear while starting or connecting the Floating License Server.

***Note**: You will find a log file "flsd.log" within the bin folder of the license server installation.*

Errors upon connecting a central License Server:

- **Error connecting to host […]**

  Please check the address and port number that has been used to specify the central license server (s. Specify Central License Server on page 42).

- **The process running on host […] is not a valid license server.**

  Upon a license request of a client machine, the central License Server on your network server tries to resolve the name of the client using "reverse lookup".

  In case that reverse lookup is failed (e.g., your server is not in the same domain/network as the client) the license server will not be able to provide a license for the client.

  In that case, you have to set the following environment variable for the central license server on your network server to solve the problem:

  ```
  FLSD_FLAGS=--no-dns
  ```

- **License key file is invalid by using NAT**

  In computer networking, the process of network address translation NAT involves re-writing the source and/or destination addresses of IP packets as they pass through a router or firewall.

  This will cause different IP addresses and the local license manager doesn't accept the license file which has been sent by the central license server upon a request of Tessy.

  To use NAT with Tessy you have to set following environment variable on the client machine:

  ```
  FLSD_FLAGS=--enable-nat
  ```

  Alternatively you may also use the following option in the configuration file of the license server: `fls.conf`

  ```
  [Server]
  EnableNat=1
  ```

Note: *You will find this configuration file for each user under C:\Documents and Settings\[Login Name]\Application Data\Razorcat\fls\config.*

Note: *You must have installed at least license server version 3.1 on your network server and Tessy v2.4.25 to be able to use this feature. The license server v3.1 will be delivered since Tessy version 2.4.25.*

Errors upon starting the License Server:

- **Error opening registry key**

  The Floating license server needs write access to the registry of Windows. This is often not allowed if you logged in with user privileges.

  This issue may appear when you want to use a node-locked license with Tessy. Normally Tessy tries to start a locale license server as application on your computer if you want to use a single license for one Tessy installation.

To start the local license server successfully with user privileges, please configure the license server as service. You need admin privileges to performing the settings. For more information on how to configure a license server as service, please refer to Install Central License Server on page 43 .

Checkout Licenses

If you want to use Tessy on computers that are (temporarily) disconnected to your network (e.g. notebooks) you may start a local license server on these computers. You can check out a temporary license from the central license server, to operate the notebooks license server independently from the network.

There are two possibilities to checkout a license:

- If you want to checkout a license for a computer currently connected to the network, please refer to section "Checkout from the License Dialog".
- If the computer is disconnected from the network (e.g. a notebook while traveling), you can checkout a license using the **Floating License Manager**. This will be described in section "Checkout using the Floating License Manager".

***Caution**: Checking out licenses for a number of days will temporarily reduce the licenses available from the central server's license pool (for the days you have chosen). There is no way to return temporarily checked out licenses!*

*You can set the environment variable "**FLSD_FLAGS=--no-keys**" on your central license server to prevent client machines from checking out temporary licenses*

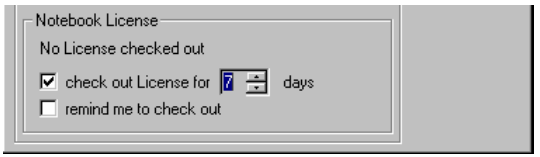
***Note:** It's further on possible to check out licenses by using the license manager installed on your central license server (see Checkout using the Floating License Manager)*

Checkout from the License Dialog

Note: *Probably this option has been disabled by your system administrator, in case that the check out option is not available. This option is available only for floating licenses (s.* *Registration on page 36).*

The easiest method to checkout a license for a few days is as follows:

- Start Tessy. Open the license dialog by clicking **Register...** in the **Help menu**
- In section **Notebook License** select option **check out License for...** and adjust the number of days (up to 30) you want to checkout the license.



- Click on **OK**.

  Tessy tries to check out a license from the central license server. The license will be saved as `notebook_like.dat` into the bin directory of the local license server. Once your notebook has disconnected from the network, you have to restart Tessy to operate independently from the central license server. As long as the license key is valid, Tessy will automatically start a local license server on this computer.

Important note*: You will check out a license for Tessy and CTE.*

Checkout using the Floating License Manager

Note: *Probably this option has been disabled by your system administrator, in case that the check out option is not available. This option is available only for floating licenses (s.* *Registration on page 36).*

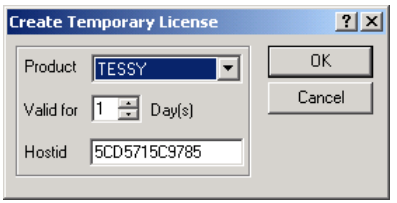
Please take the following steps to checkout a license from the **Floating License Manager**:

- Start the **Floating License Manager.** You will find the license manager in the folder **Tessy** within the Windows Start Menu.

- Click on **Check out** in the toolbar
- The following dialog will pop up. Please select the product to check out and the desired number of days (up to 30).

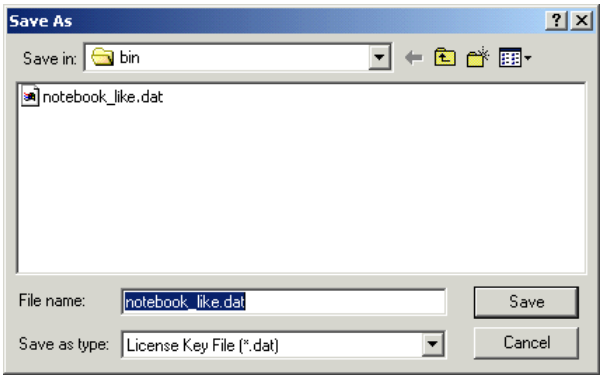
Important note*: The License Manager will check out a license for Tessy* ***and*** *CTE, if you choose* ***TESSY!***

The edit field **Hostid** contains the HostID of the local computer.



Note*: If you want to check out a license for a computer disconnected to the network (e.g. for a traveling colleague, whose license has expired), then you need to enter the Host ID of this computer into the* Hostid *field.*

- Click on **OK.** The following dialog will appear:



Save the generated license file to your local disk. The file name must be notebook_like.dat.

Caution: *Don't cancel the dialog before saving the license key into a file. You will lose the generated key. Therefore, FLM will also copy the license string into the main window of FLM (e.g., Checked out License e79abb53dc19…). You must then copy-n-paste the license key into a text file "notebook_like.dat" manually. Otherwise you will lose temporarily one license from the central server's license pool for the days you have chosen. There is no way to return temporarily checked out licenses!*

- Transfer the `notebook_like.dat` to the computer, where you want to run Tessy locally. Save the license file into the bin directory of the license server, normally to `C:\Program Files\Razorcat\fls\bin\`
  (adjust the path according to your installation directory).

Note: *If you cannot transfer the license key file directly to the target computer, then just type the license key (the first line of the license key file) in a file named* `notebook_like.dat.` *Save this file on the target computer into the server´s bin directory, normally to* `C:\Program files\Razorcat\fls\bin.`

Start Tessy on the target computer. As long as the license key is valid, Tessy will automatically start a local license server on this computer.

Quickstart

Tutorial

The purpose of this tutorial is to give you a “quick start” understanding of the features incorporated into Tessy.

The basic functionality will be shown on a sample test cycle using a simple source code example (`is_line_covered_by_rectangle`). After this short introduction, we would like to encourage you to discover further features of Tessy using our sample project database. Please choose **Samples** from the **Help** menu to open that database. This file will be copied to `c:\tessy\samples\[version]\sample.pdb` by default. The path may differ depending of your installation.

***Note**: Tessy includes the GNU gcc compiler for Windows and can run the tests on the Windows PC (which is normally only acting as host). The example of the tutorial using the GNU tool set instead of an embedded compiler. You may change the default target setting to use your target compiler (s. Tessy Environment Editor). We recommend to use gcc at first.*

Upon successful completion of the tutorial you will be able to:

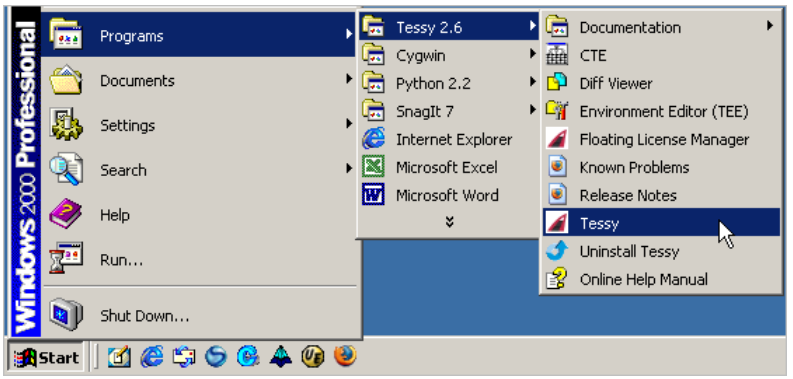
- Create a **Database** that contains all meta data belonging to your projects.
- Create a **Project** and add a **Module** for the given source code.
- Add source code for testing**.**

- Use the Test Interface Editor **TIE** and Test Data Editor **TDE** to specify test data and expected results for the test object under testing.
- Execute a test run and evaluate the test results using reports.

Start Tessy and Create a New Database

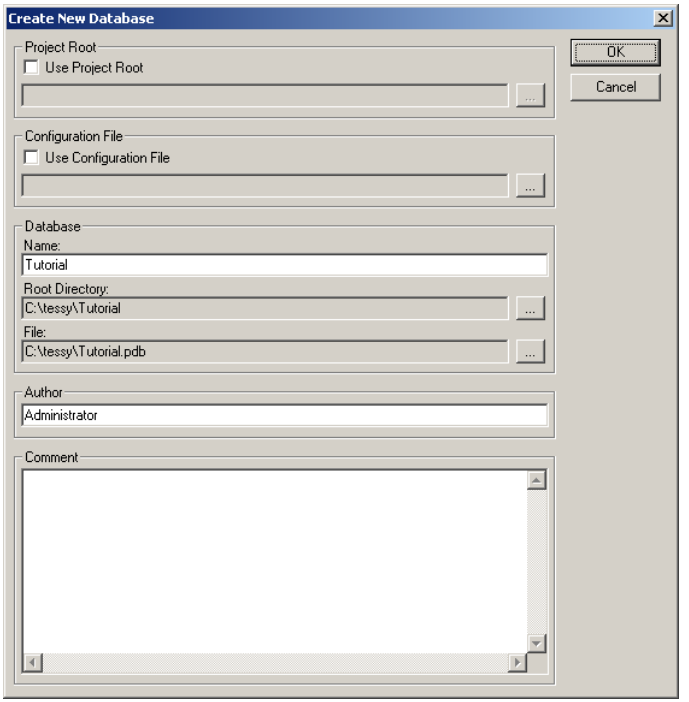
Open the Tessy program:

- Select **Programs|Tessy 2.6|Tessy** from the Windows Start menu or use `[Ctrl]+[ALT]+[T]`. The program will start with an empty main window.



Create a database:

- Please select **File|New Database** or click on the respective icon in the toolbar to create a new database.
- In the following dialog, please enter a **Database Name** for the new database, e.g. **`Tutorial`**.



- While typing, Tessy will complete the paths for **Root Directory** and for **Database File** automatically. We suggest to accept the default settings. Root Directory will hold all files generated by Tessy belonging to your projects and modules. The database file contains all Meta data about projects, modules and test objects.
- Once you have completed your input, click **OK**.

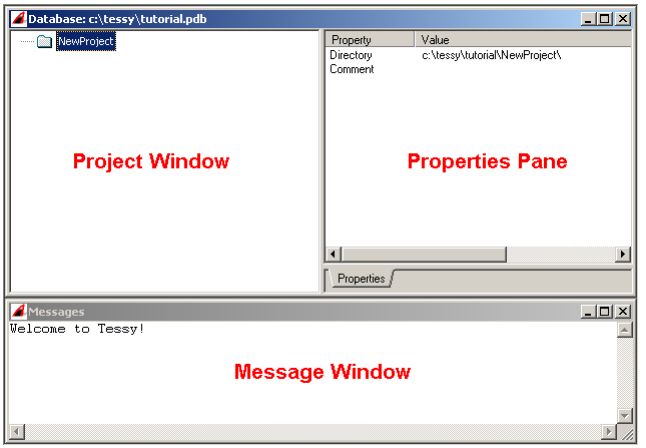
***Note**: The **Project Root** allows to specify a root directory of your development projects, so that all paths (e.g. sources, includes, etc.) may be related to this root (s. The Project Root). The **Configuration** option allows to select a specific configuration file for the project database which contains only your target environments you want to use. If no configuration file has been chosen for a given database, Tessy will use the global setting provided by the Environment Editor. For our sample project, we don't need to use both features. Therefore deselect please the respective check-box.*

Now, you have finished the first step:

A new database has been created and a new project has already been added into the database by default.

Tessy will open your database within a window containing two areas:

- The left window holds the project and module hierarchy.
- The right pane contains the properties of a selected project or module.

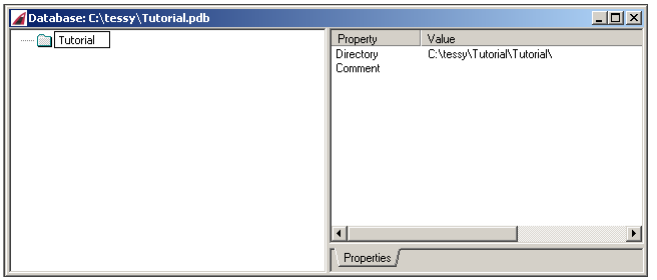


- Additional, at the bottom of this window you will also see the messages window of Tessy.

Create a Project, a Module

While creating the database, a new **Project** folder called *New Project* has already been created. You may create another project folder using **Project|New…**

- Please change the name to **`Tutorial`**

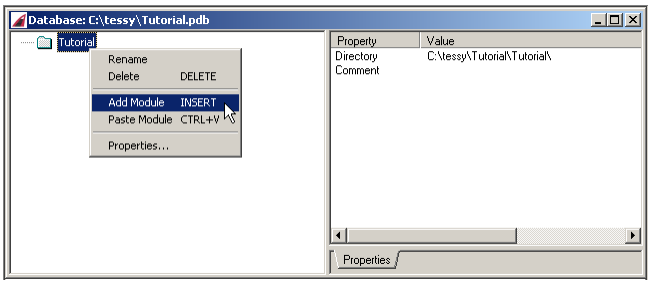


---

*You may change project or module names by clicking twice on the name with a slight delay or by selecting **Rename** from the context menu*

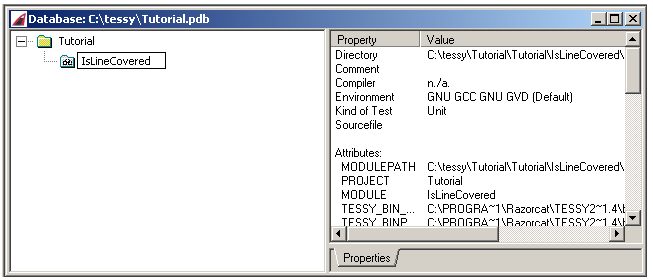
Please add a new module to this project folder:

- Select the project folder and choose **Add Module** from the context menu or just press `[Insert]`.



The color of the project icon will change to yellow indicating that it now contains a module.

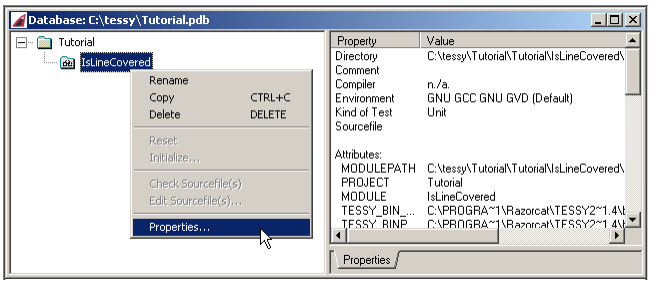
- Please change the name of the module to **`IsLineCovered`**.



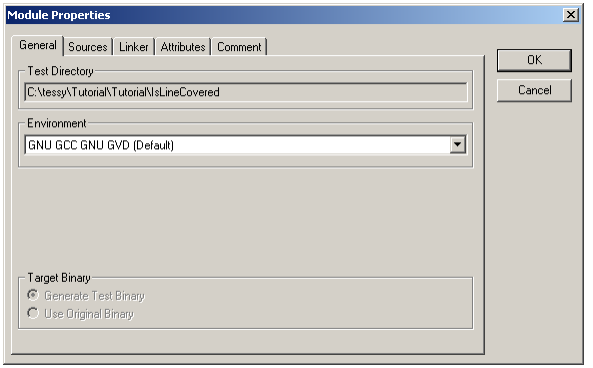
*You may change project or module names by clicking twice on the name with a slight delay or by selecting **Rename** from the context menu*

Specify Target Environment

- Select the newly added module **`IsLineCovered`** and choose **Properties** from the context menu.



The *Module Properties Dialog* will pop up.



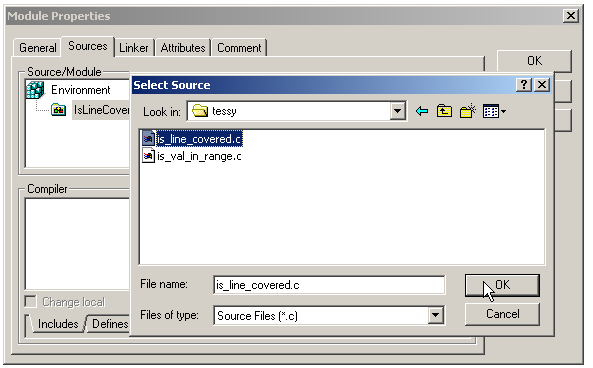
In this dialog, you will enter general settings for the module, e.g. target environment, source files, additional compiler options, etc.

Within the **Genereal** tab of the dialog we have to specify the **Target Environment** which we want to use. We will use the GNU tool set instead of an embedded compiler. You may change the default target setting to use your target compiler. We recommend to use the GNU tool set for the tutorial at first.

Note: *Potentially, your Target Environment is not yet available in the Environment list box. In that case you have to enable and configure your environment (Compiler/Debugger) using the Tessy Environment Editor.*

Add Source File

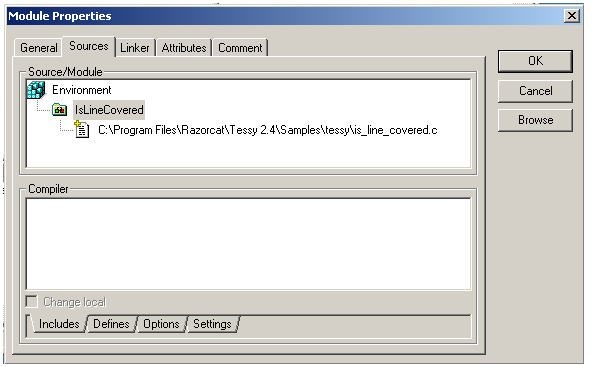
- To enter source files for the module, please select the **Sources** tab and click on **Browse.** *A file selection dialog will appear.*



- Please select following file and click on **OK**:
  `C:\Program files\Razorcat\Tessy_2.6\samples\tessy\is_line_covered.c`

***Note**: If you have installed Tessy in a directory other than the default, you will have to change the path "C:\Program files\Razorcat\Tessy_2.6\" accordingly.*

You will now see the selected source file in the **Source/Module** pane. At the moment, we don't need to use the other tabs **Linker** and **Attributes** nor the special tabs **Includes**, **Defines**, **Options** and **Settings** (s. Module Properties).



- Click **OK** to close the dialog.

Open the Module

- Double click on module **`IsLineCovered`** or click the plus sign.

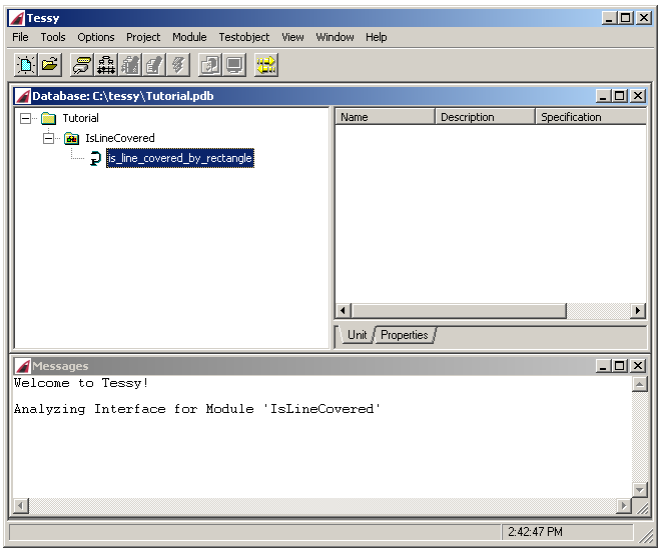
  Tessy will preprocess and analyze the source file(s). The module will open if no errors appeared.

  After this processing, all functions which were defined in the source file(s) are displayed as children of the module folder.

  Our sample source file contains only one function, our test object

  `is_line_covered_by_rectangle`

***Note**: The term **Test Object(s)** will define the function(s) displayed in the module folder and we are attempting to test. Test object is used throughout the manual.*



If the **test object** `is_line_covered_by_rectangle` is selected, the **Unit** and **Properties** tab appear in the right pane of Tessy's main window.

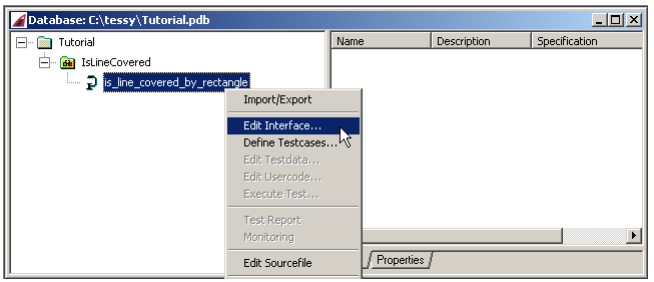
- The **Properties** tab lists all settings, which has been chosen for the module, e.g. compiler, debugger, list of source files.
- The **Unit** tab holds the **test cases** of the **test object** and is empty at the moment.

Edit Test Object Interface

In this section you will review the interface settings of the test object and determine, which values are input and which ones are output variables.

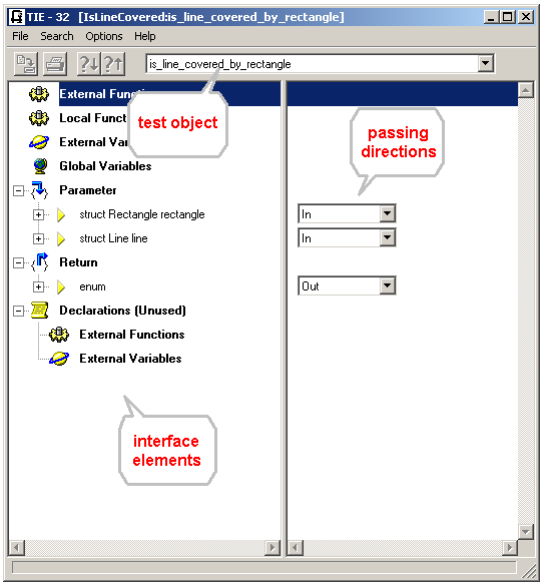
Note: *Upon opening the module, Tessy will try to analyze all interface objects of your test object automatically. In case that no "unresolved object" has been found, Tessy will open the respective "clamp" (the test object icon) to indicate that. You may change these default interface settings to your needs.*

- Click on test object `is_line_covered_by_rectangle` and select **Edit Interface** from the context menu.



- The **Test Interface Editor** TIE will open (see next picture below).

  Now, you can see all used variables and called functions of the test object (if any).



- On the right side, there is a column with combo boxes indicating the passing directions of the respective variables.

  The variables are either read within the function (**`In`**), written within the function (**`Out`**), both read and written (**`InOut`**), to be altered by Usercode (**`Extern`**), or they are simply not used within the function (**`Irrelevant`**). Upon opening the module, Tessy will try to set the default passing directions automatically (You may change these default interface settings to your needs).

Please have a look at the several sections on the left side of **TIE**:

- **Section: External Functions**
  In this section, you will find all called functions of the test object. These functions are not defined within the source file(s) of the module.
- **Section: Local Functions**
  In this section, you will find all functions defined within the source file(s) and called from the test object.
- **Sections: Global / External Variables**
  The next two sections list the external and the global variables. The **External**

Variables are all used variables, which are not defined within the source file(s) of the module.

- **Section: Parameter**
  In this section, you will see the **Parameter** of the test object. Our sample test object has two structures: `Line` and `Rectangle`.

  You can browse through these structures by clicking on the plus sign to see the basic components.

- **Section: Return**
  In this section, you will see the **Return** value of the test object. In our case defines an `enum variable` values for `"yes"` or `"no"`.

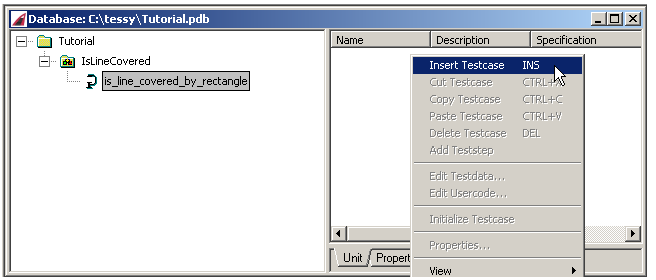
- **Section: Declarations (Unused)**
  At the end of the list, you will see the section **Declarations (Unused)**. These sections will be explained in more detail later in this manual. For our sample, we don't have to care about these sections.

Please exit the **TIE** by pressing [ALT]+[F4]or use **EXIT** from the **File** menu. Since we didn't change anything, we don't have to save something.

Enter Input and Expected Values

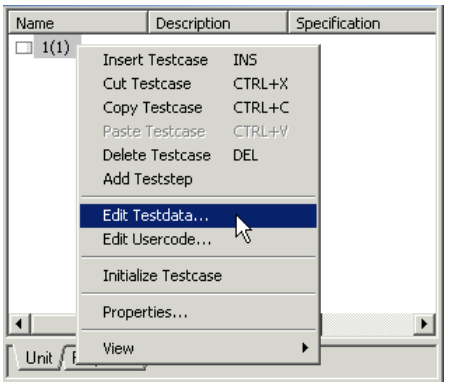
In the next step of our tutorial we need to create at least one **test case** to enter input and expected values of our test data. Please do the following:

- Click in the right pane of the main window (**Unit** tab) and choose **Insert Testcase** from the context menu.

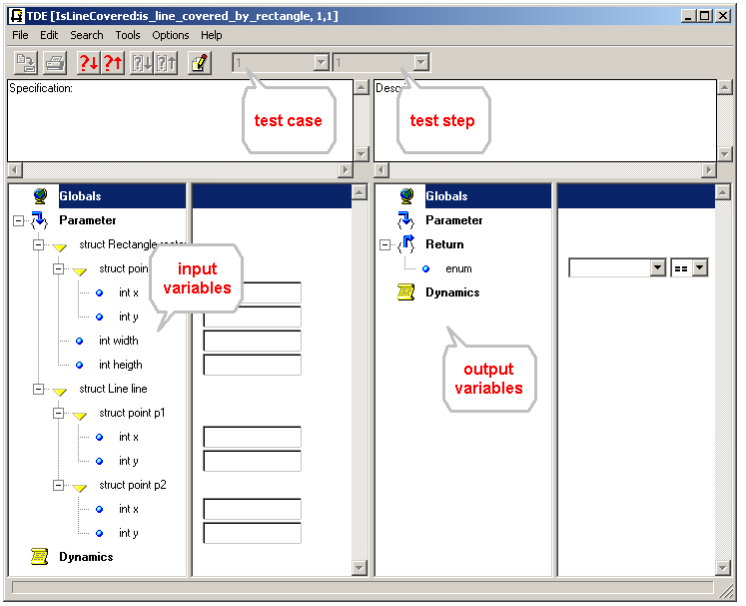


A new test case will be created in the list. One test case includes at least one test step to hold the test data for test execution. You can add additional test steps to each test case. Every test step contains a complete set of test data (s. chapter "Enter Test Data > Test Steps").

- Now, please select the test case and choose **Edit Testdata** from the context menu.



*Test Data Editor **TDE** will open.*



The main lower window of the **TDE** is vertically split into two areas:

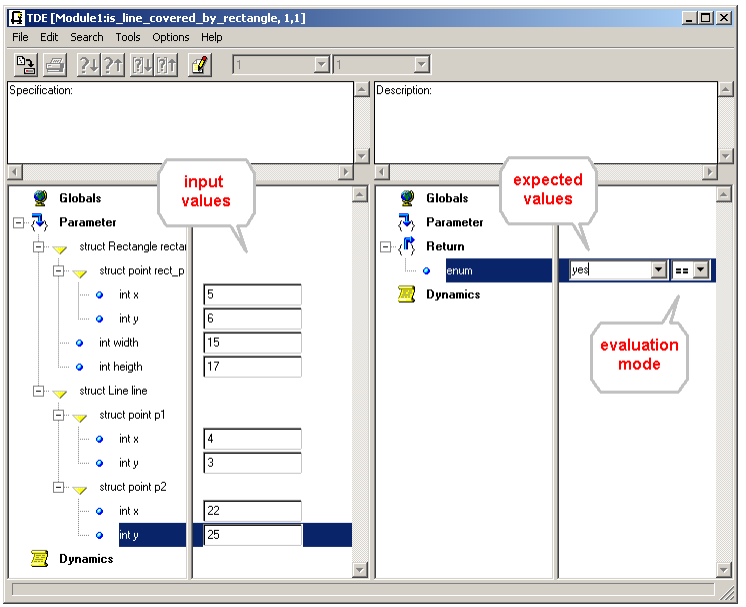
- On the left side you will see the input variables (**IN**)
- On the right side you will see the output variables (**OUT**).

If you have more than one test case respectively test steps created, you can use the combo boxes to switch to another one.

Specify Input Values:

To enter appropriate test data for our sample test object, please use following input values from the picture below:

You can browse the interface down to the basic components or click on **Next Undefined Value** that will guide you automatically to the next variable where you have to enter an input value.



About the test object: `is_line_covered_by_rectangle`

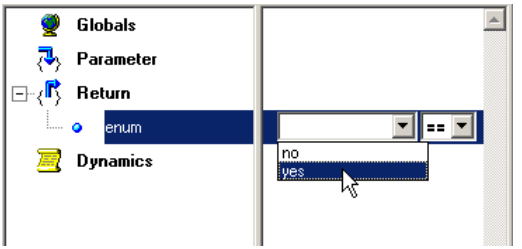
The structure **`line`** consists of the points **`p1`** und **`p2`.** The structure **`rectangle`** has a starting point and the size of a rectangle **`width`** and **`height`**.

The function **`is_line_covered_by_rectangle`** checks, if the line is covered (partly or completely) by the rectangle.

Specify Output or Expected Values:

Now, you have to specify the output (or expected) values of the test object. In the right pane of the **TDE**, you can enter the value that will be returned by the test object.

In our case, we expect that the line will be inside of the rectangle. That is, we need only to choose "`yes`" from the combo box of the return value (see next picture below).

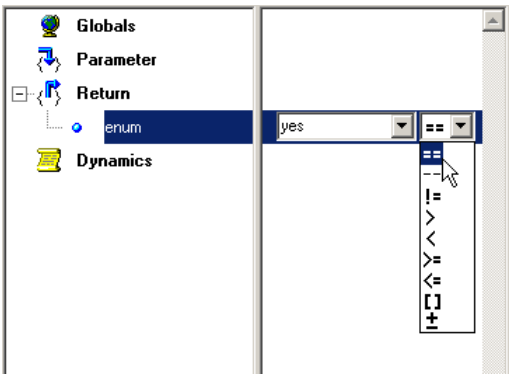


Specify the Evaluation Mode

Now, you have to specify the evaluation mode for the "expected" value. This defines how to compare the **Expected Values** with the **Actual Values** resulting from the test run.

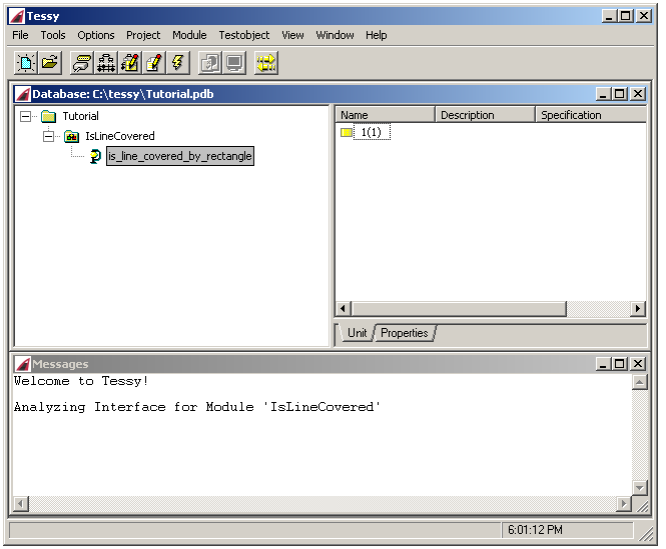
- Please use the right-most combo box to specify the evaluation mode.

  You can choose one of the predefined logical operators from the dropdown list: In our case, choose **==** for equal.



- Click on **Save** to exit TDE.

Please note, that both the test object icon and the test case icon will be turned to **yellow**. This will indicate that we are now ready to run the test.



Execute Test Run

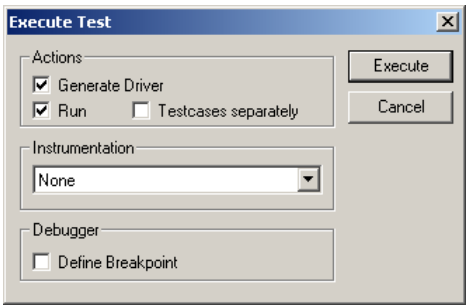
All necessary preparations are done at this point, so we can start the test run:

Click on **Execute Test** in the toolbar or choose the context menu of the test object.

In the following dialog, you may select the actions to be performed during the test execution.

- Please use the default selections shown in the picture below:

  Actions: Generate Driver and Run
  Instrumentation: None

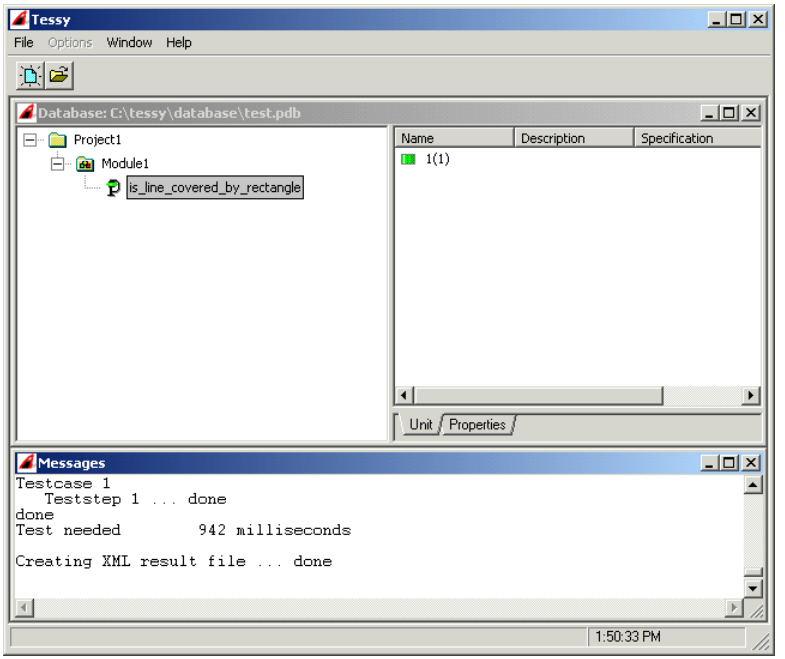


- Click on **Execute** to execute the test.

Note: *You have to select at least one test case to run the test. Otherwise the Run check box will be disabled. If you tick the option "Define Breakpoint", the GNU Visual Debugger will appear. It's then possible to debug the test object in question.*

An **Execute Test** dialog will be shown while Tessy generates, compiles and links the test driver and runs the test. This may take some time. You will see the progress of the test execution within the message window of Tessy. If any errors occur, these messages would be printed there.

At the end of the test run, both the test object icon and the test case icon should be turned to **green** (see picture below). If not, the expected value didn't fit to the actual value.


Tessy
File
Options
Window
Help
Database: C:\tessy\database\test.pdb
Project1
Module1
is_line_covered_by_rectangle
Name
Description
Specification
1(1)
Unit
Properties
Messages
Testcase 1
Teststep 1 ... done
done
Test needed 942 milliseconds
Creating XML result file ... done
1:50:33 PM

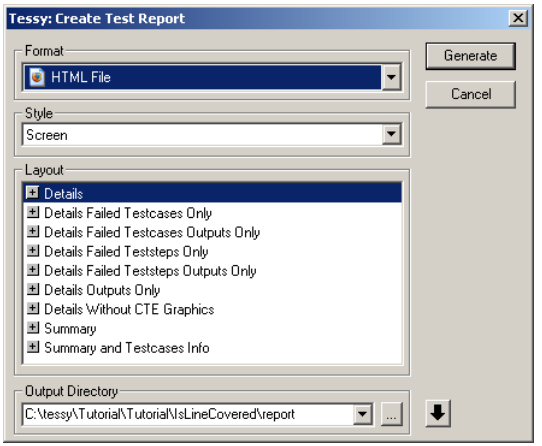
Evaluate Test

The last step of the tutorial is to evaluate the test run. For this purpose, you can create a test report or simply open the **TDE**.

Create a Report

Click **Test Report** either from the task bar or choose the context menu of the test object.

The following dialog will appear:



Please choose following parameters:

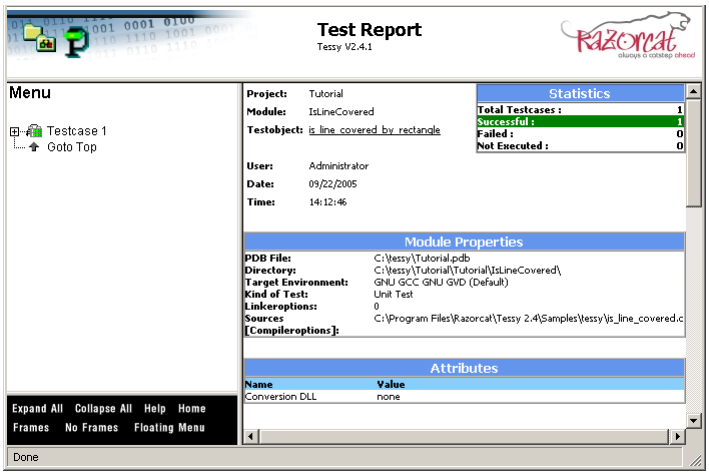
- **Format:** HTML Document
- **Style:** Print
- **Layout:** Details
- **Output Directory:** Please specify an output directory

If you have finished your selection, please click **Generate**. Now, the test report will be created and shown in your standard Web Browser (see picture below).

Main components of the test report

The right hand window of the test report contains following sections:

- In the upper left corner, general settings are listed, like project name, module name or the name of the test object.
- Next to it, you see a summary of all test cases, sorted by successful and failed.
- In the tables below, you see the module properties and module attributes listed (if any).



Detailed information about test cases

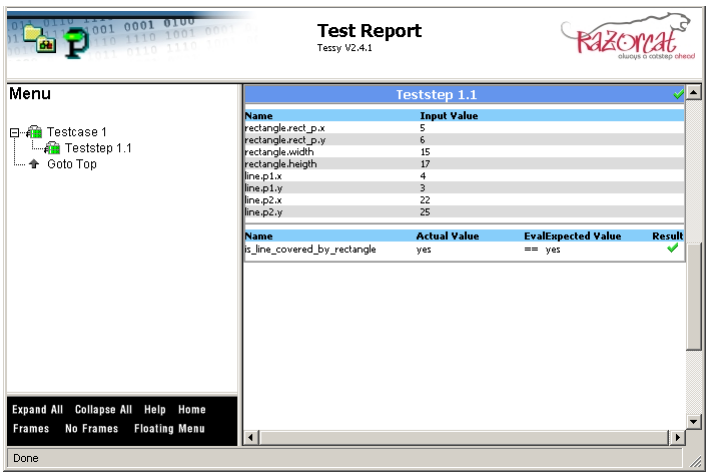
***Note**: The **test case** is a container of **test steps** and contains at least one **test step**.*

The next picture shows the summary of each test case:

- Input values {`5,6,...,22,25`}
- Actual values {`yes`}
- Expected values {`yes`}
- Result of the test run.

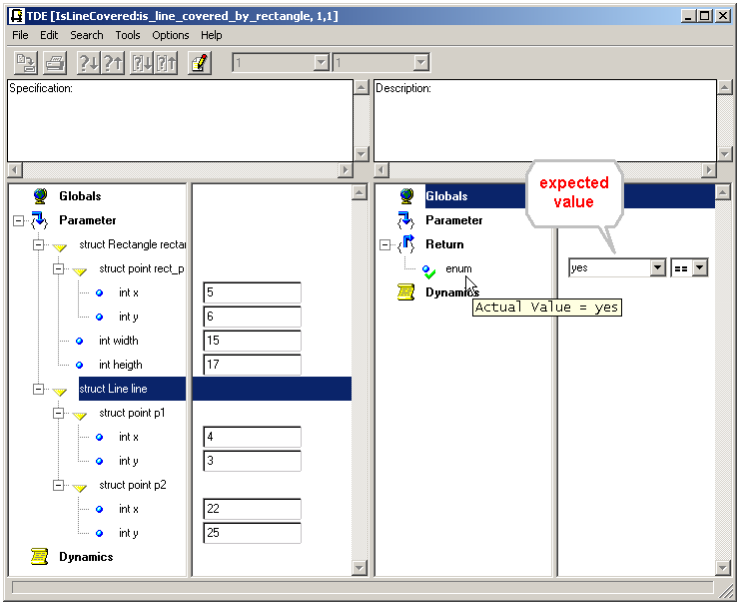
  Specification: The actual value after test run {`yes`} must be equal {`==`} with the expected value {`yes`}.

  Result: As the actual value and the expected value are equal, the result of the test is ✔.



Open TDE

- Double-click the **test case** or choose **Edit Testdata…** from the context menu. *The TDE will open.*



Taking into account the evaluation mode (**==**), a test was successfully executed, if the expected values specified in TDE does coincide with the actual values. The TDE will indicate that using a green tick (✔).

- To display the **Actual Value** of **enum**, please move the mouse pointer over the return value **enum**, as shown in the picture above.

✔ **Congratulations, you have completed the Tessy tutorial!**

***Note**: You will find more information on special features and dialogs in the following chapters of this manual.*

You may also review our sample project database. For this purpose, please choose

Samples *from the* ***Help*** *menu to open the database. This file will be copied to "*`c:\tessy\samples\[tessy version]\sample.pdb`*" by default. The path may differ depending of your installation.*

Test Organization

Databases

Tessy manages its projects by means of a project database. By default, Tessy project databases have a **.pdb** extension.

The **PDB** contain meta data of projects (s. Creating a Project) and modules (s. Creating a Module) as well as information on the test environment, used source files, compiler, debugger, etc.

Upon creating and opening of a module Tessy generates two additional database files to store data necessary for the test run:

- Interface database **.idb** (contains the module interface information).
- Test database **.tdb** (contains your test data).

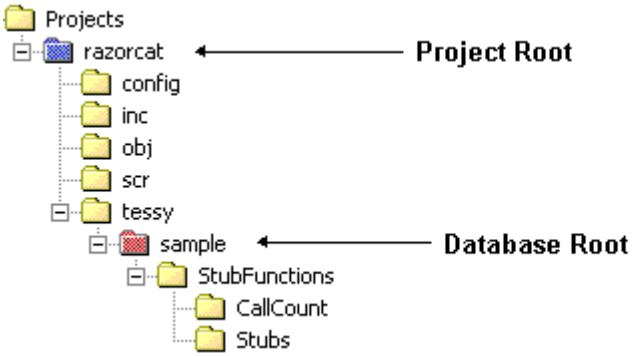
For more information see chapters "Files in the Module Directory" and "Version Control".

The Project Root

Tessy provides a feature that allows you to specify a so called **Project Root.** The **Project Root** specifies a root directory of your development projects, so that all paths (e.g. sources, includes, etc.) may be related to this root.

This may give you the possibility to transfer your development projects to another installation. Every of your Tessy project databases will have their own project root specified within the Registry of Windows. Tessy will ask you for this root directory if you open the project database for the first time.

When you create a new Tessy project database, e.g. `sample.pdb`, Tessy will store the database file in the specified project root (`$(PROJECTROOT)\tessy`) and creates in parallel a project folder with the same name (by default, in a folder named "Tessy" where the database file and the database root folder are stored (s. Creating a Project Database). Initially, this project folder of the project database will be the **Database Root** directory.



This database root directory will contain all project folders (StubFunctions) and their corresponding module folders (CallCount, Stubs) which have been created in Tessy for the given database, e.g.

`$(PROJECTROOT)\tessy\sample\StubFunctions\Stubs\`

The module folder `Stubs` will contain all files used to compile and execute the test objects (e.g. AddToMemory), as well temporary generated files created by the compiler (see Save a Module). ´

***Note**: Please refer also to the application notes "Test Database Setup" from the* **Help|Documents** *menu for more information to share Tessy test databases and test projects between development teams.*

Creating a Project Database

We would recommend using the so called **`PROJECTROOT`** (s. The Project Root) and a **`custom configuration file`** (s. Creating Configuration Files) for your project database.

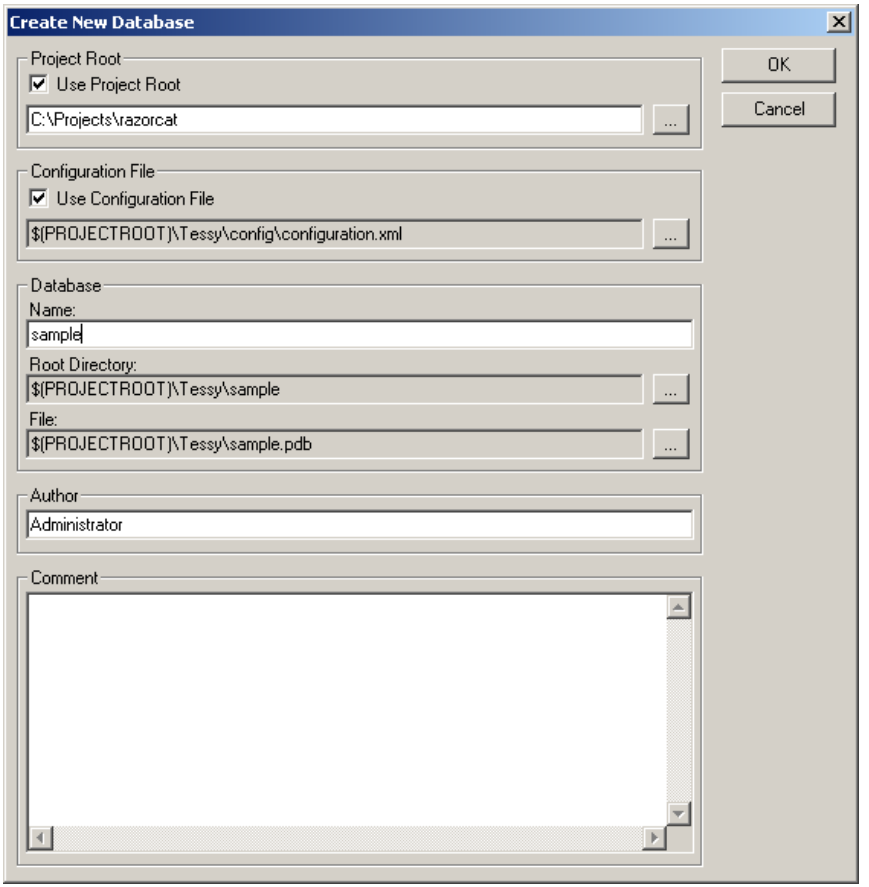
All paths for e.g. source files, includes and other project files will be related to this project root. The custom configuration file contains appropriate target environments for the project database which can be defined with the environment editor (s. TEE).

Tessy will create a folder named **`Tessy`** to store files related to this project data base if a **`PROJECTROOT`** has been specified (s. below). The configuration file will also be stored in a special folder, named **`config`** which will be created within the **`Tessy`** folder (s. below).

Steps to Perform

To create a new project database either click on **New Database** in the **File** menu, or click the respective icon in the toolbar.

The *Create New Database* dialog will open.



Specify Project Root

By default, you have to specify the project root of your Tessy project (s. The Project Root).

If you don't want to use a project root, please deselect the check-box (not recommended).

- To specify a **Project Root,** click the path selection button and choose an appropriate directory from the dialog.

Specify Configuration File

By default, you have to specify a custom configuration file for the project database. You may use a specific configuration file for each project database (s. Creating Configuration Files). This file should be stored in the recommended folder, named **`config`** within the **Project Root**.

The **Configuration File** will automatically be created, if it doesn't already exist. It will contain the GNU/gcc compiler environment that is enabled by default when installing Tessy.

If you don't want to use a custom configuration file, please deselect the check-box (not recommended).

- To select a **Configuration File,** click the path selection button and choose an appropriate file for your database.

Specify Database Name

- Enter a database **Name**. Tessy will complete the database **Root Directory** and **Database File** name automatically. In case you have specified a project root, Tessy creates also a folder named **`Tessy`** where the database file and the database root directory are stored (otherwise a default path is used, e.g. **`c:\tessy`**).

Specify Database Root Directory

By default, the database **Root Directory** will be inserted with the input of a database **Name** and should not be changed. You may change this directory as follows:

- To specify a database **Root Directory**, click the path selection button and choose an appropriate directory from the dialog.

Specify Database File

By default, the database **File** will be inserted with the input of a database **Name** and should not be changed. This file will be stored in the **`Tessy`** folder if a **`PROJECTROOT`** has been specified.

You may change this directory as follows:

- To specify a database File, click the path selection button and choose an appropriate directory from the dialog.

Specify Author

- Enter the Author of the database. The Windows login name is used by default.

Specify Comment

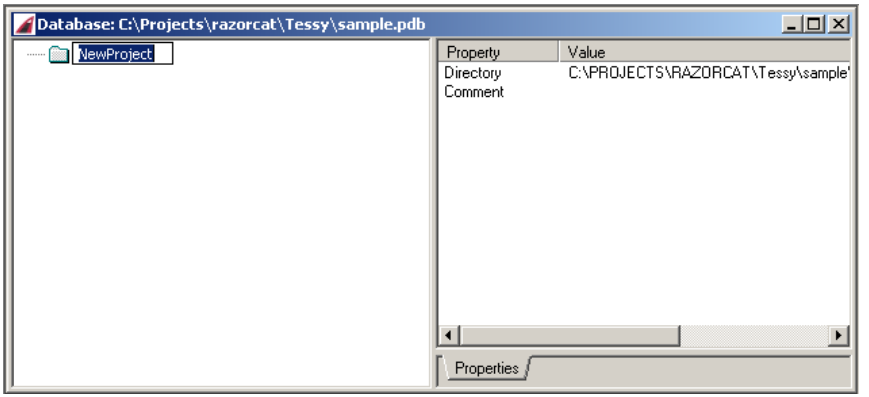
- Enter a Comment. This may help you to distinguish your project databases.

Save Your Settings

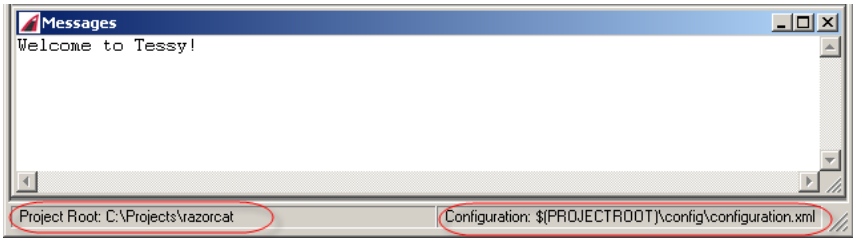
Click **OK** when all settings are complete. Tessy will now create

- a new PDB file
- a new configuration file (if not existing)
- the database root directory

Tessy will also open the database. A *NewProject* folder (see: "Creating a Project" on page 91) has been already added (see next picture).



The current project root / configuration file setting is displayed within the status bar of Tessy.



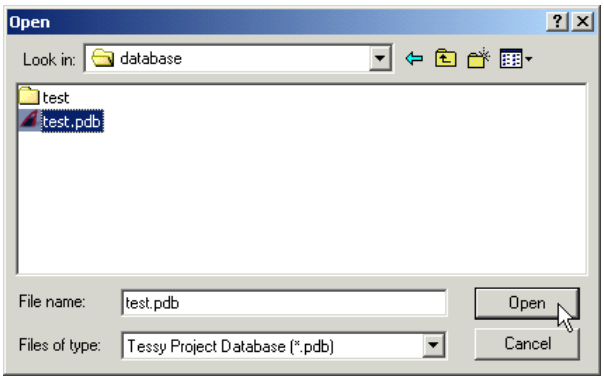
***Note**: Please refer also to the application notes "Test Database Setup" from the* **Help|Documents** *menu for more information to share Tessy test databases and test projects between development teams.*

Open a Database

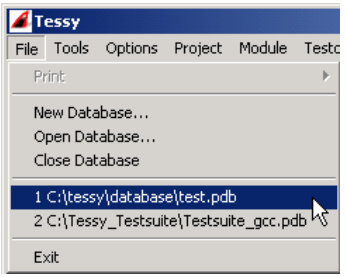
To open an existing database choose **Open Database** in the **File** menu, or click the respective icon Open Database in the toolbar.

The *Open* dialog will pop-up.

Please select a database file and click **Open**.



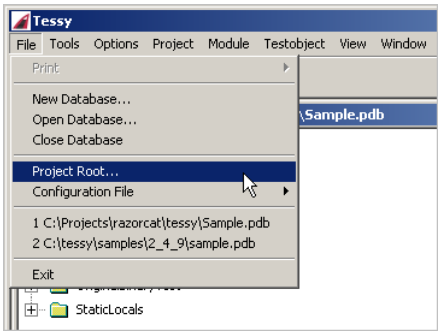
Alternatively, you can select the last used database from the **File** menu.



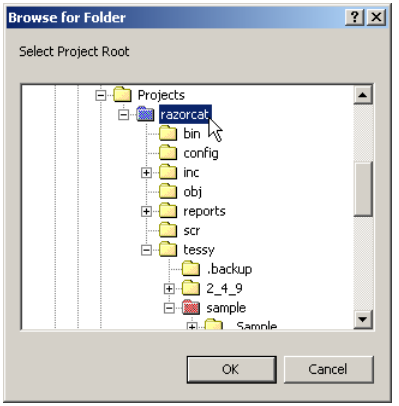
Assign a Project Root

The project root is an optional setting (s. The Project Root). You may specify a root directory of your development projects. All paths (e.g. sources, includes, etc.) will be related to this root. If not already specified (see Creating a Project Database), you may assign a ProjectRoot for each project database as follows:

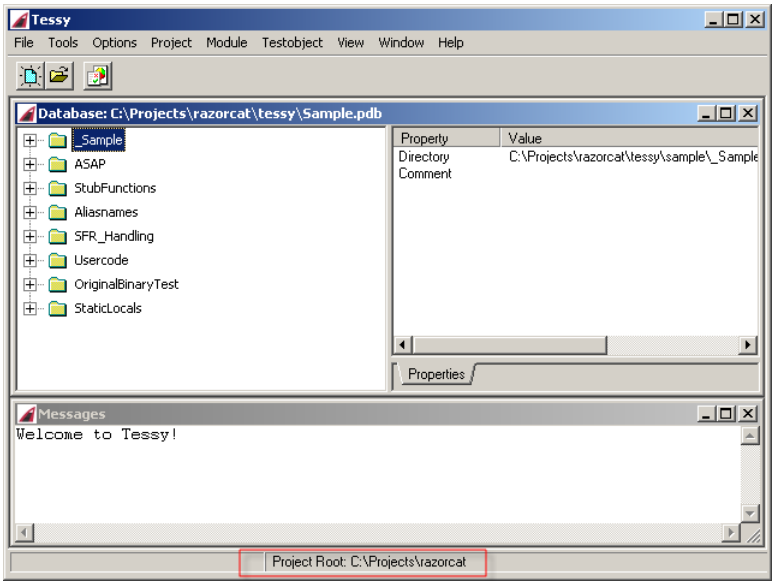
- Open a project database.
- Choose **Project Root**… from the **File** menu.



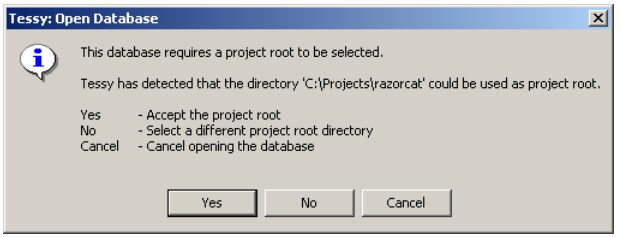
- Select an appropriate path and click OK.



Tessy stores the Project Root for each database in the Registry of Windows. The current **Project Root** will be shown in the status bar of Tessy.



If you open this database on another PC once more, Tessy will ask you for the Project Root automatically.


Tessy: Open Database
This database requires a project root to be selected.
Tessy has detected that the directory 'C:\Projects\razorcat' could be used as project root.
Yes - Accept the project root
No - Select a different project root directory
Cancel - Cancel opening the database
Yes
No
Cancel

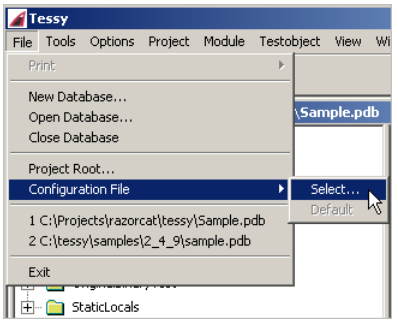
This assumes that you have already installed your project files.

Assign a Configuration File

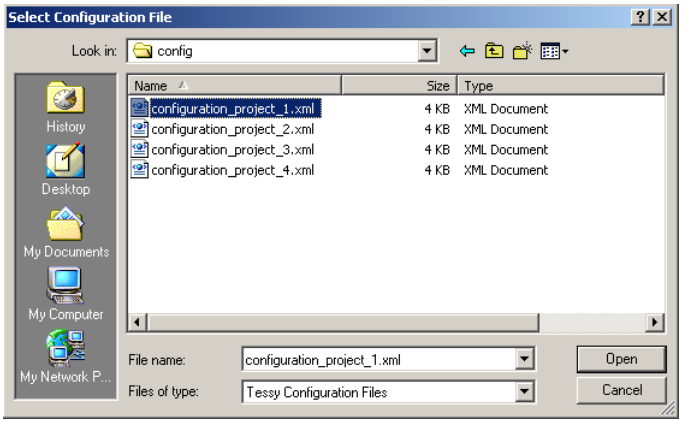
You may assign a specific configuration file for each project database (see Creating Configuration Files), This file contains only your target environments you want to use.

To use a configuration file:

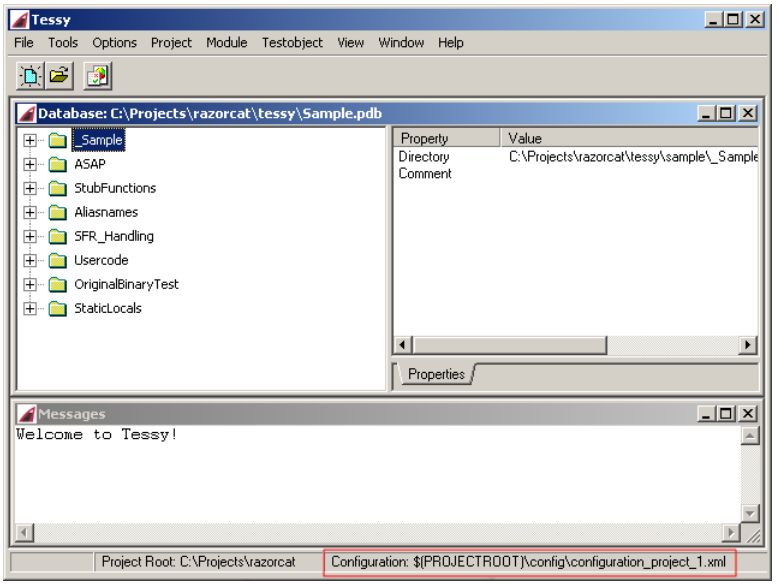
- Open a project database.
- Choose **Configuration File|Select**… from the **File** menu.



- Select an appropriate configuration file and click **OK**.



Tessy stores the path to the assigned configuration file for each database in the Registry of Windows. The used configuration will be shown in the status bar of Tessy.



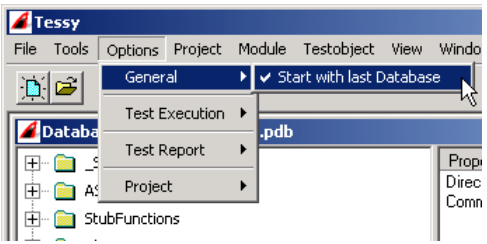
***Note**: If you open this database on another PC once more, Tessy will find the appropriate configuration file automatically. This assumes that you have already installed your project files.*

If no configuration file has been chosen for a given database, Tessy will use the global setting provided by the Environment Editor:

```
configuration: default
```

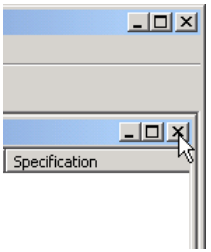
Start Tessy with Last Used Database

If you want Tessy to open the last used database upon startup, choose **Start with last Database** from the **Options|General** menu.



Close a Database

To close the database click on **Close** in the **File** menu or just close the project window.



Delete a Database

To delete a database you can just delete the database file (*database_name.pdb*) and their corresponding database **Root Folder** using the Windows Explorer. Please make sure that this database is really not needed anymore.

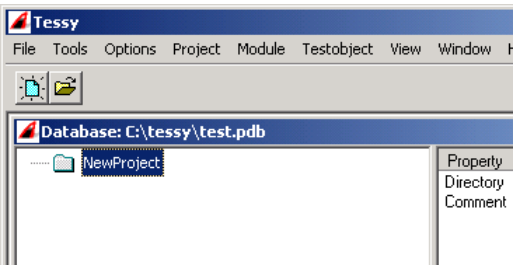
Creating a Project

A **Project** is a folder used for the test organization and may contain one or more **Modules** (see Creating a Module).

Add a Project

- To create a new project folder choose **New** from the **Project** menu.

A **NewProject** folder will appear in the project window.

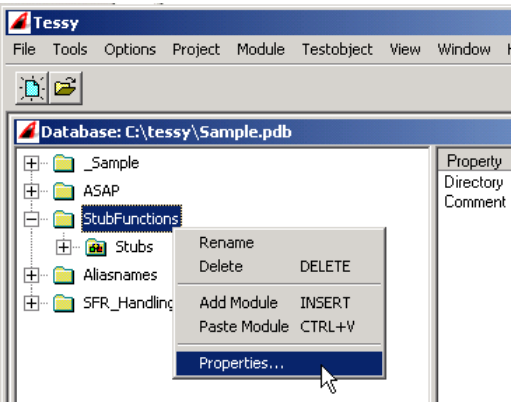


Note: *A valid name cannot contain any of the following characters: blanks / \ " ~ ! @ # $ % & * ? = [ ] { } < > . ; ' ` | Only simple brackets ( ) , dashes -, and underscore characters_ are permitted!*

Comments on the Project

If you have more than one projects created, it may be useful to add a short description for every project. This can be done as follows:

- Select the project folder and choose **Properties** from the context menu.



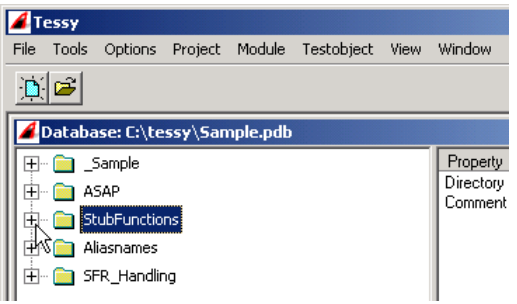
The *Project Properties* dialog will open

- Enter or read a short description on the project. Click **OK**.

Open a Project

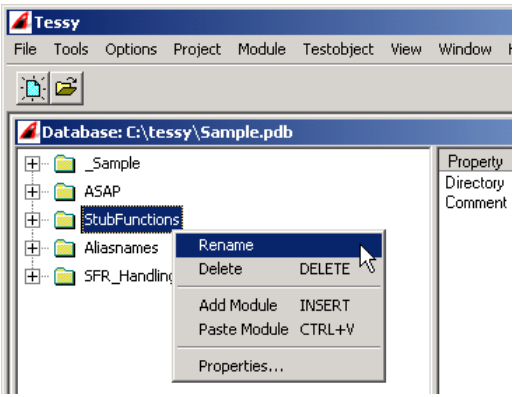
You can only open a project if already modules have been added (see Creating a Module).

- Click the ⊞ or double click the project name. The project folder will open and the modules will become visible.



Rename a Project

- To rename a project you may either click once the project name or select **Rename** from the context menu.



Note: *A valid name cannot contain any of the following characters: blanks / \ " ~ ! @ # $ % & * ? = [ ] { } < > . ; ' ` | Only simple brackets ( ) , dashes -, and underscore characters_ are permitted!*

Delete a Project

To delete a project, do the following:

- Select a project you want to delete.
- Choose **Delete** from the context menu or use `[Del]` on the keyboard.
- Click **OK**.

Tessy will remove the project folder and all containing module folders. This will also delete the project folder from the hard disk.

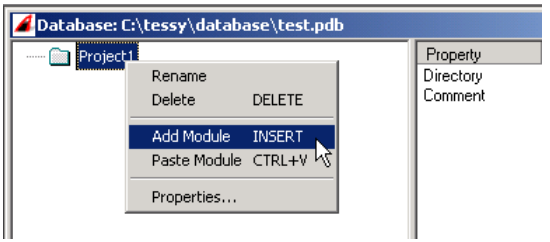
Creating a Module

A Module contains all C functions (test objects) derived from your source file(s) and specifies settings needed for the test. You can only create a module in a project folder (see Creating a Project).

Add a Module

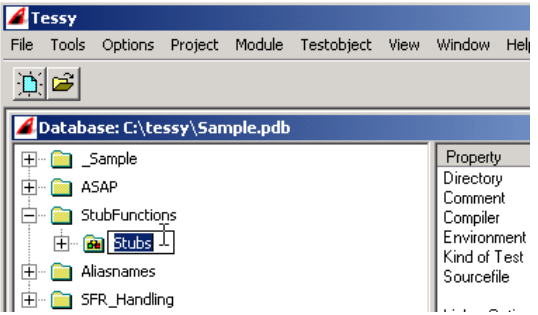
To create a new module, do the following:

- Select the respective project and choose **Add Module** from the context menu or press [INS] on the keyboard. *A new module folder will be added.*



Rename a Module

- To rename a module folder you may either click once the module name or select **Rename** from the context menu.

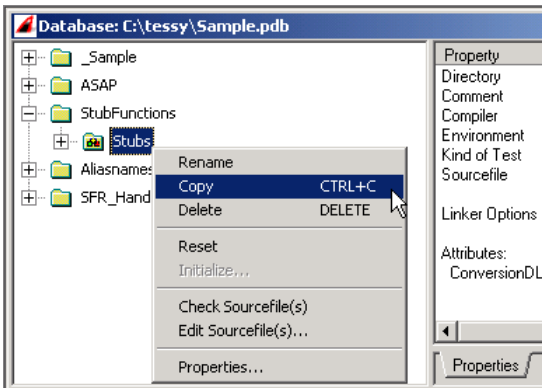


Note: *A valid name cannot contain any of the following characters: blanks / \ " ~ ! @ # $ % & * ? = [ ] { } < > . ; ' ` | Only simple brackets ( ) , dashes -, and underscore characters_ are permitted!*

Copy and Paste Modules

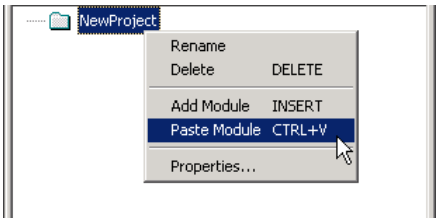
For the same open database, you may copy a module from one project folder to another one. Please do the following:

- Select the module that you want to copy.
- Choose **Copy** from the context menu or use `[Ctrl]+[C]`. The module will be copied.



To add the copied module to another project folder do the following:

- Select a project folder and choose **Paste Module** from the context menu or use `[Ctrl]+[V]`. *The module will be inserted.*

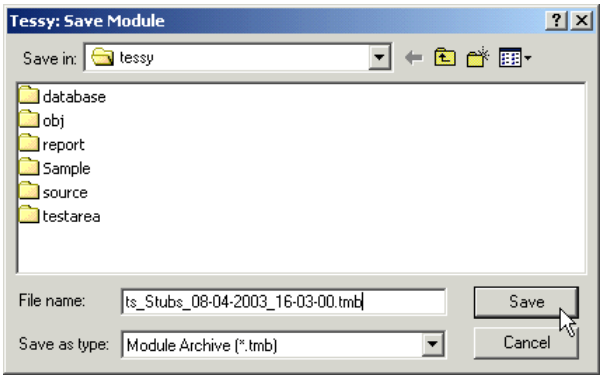


Save a Module

You can archive all settings and the test data of a module in a file. This is recommended if you want to change your test conditions (e.g. regression testing). In case of problems you may reject all changes by restoring the module (s. Restore a Module).

Do the following:

- Select a module and choose **Save** from the **Module** menu.
  The *Save Module* dialog will open.



- Specify a directory and a file name. Please use the extension `*.tmb` or accept the default name (recommended).

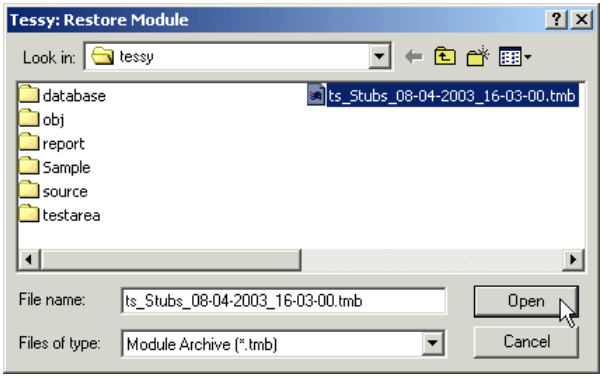
The **TMB** file will store following information:

[module name].idb	Interface database, contains the module interface information.
[module name].tdb	Test database, contains your test data.
*.tpl	Template files, containing Usercode and code of stub functions.
*.cte	CTE document.
*.xml	Test result file.
Archive.header	Information about the test environment.

Restore a Module

You can restore a module from an existing `*.tmb` file into any project folder. Do the following:

- Select a project folder in Tessy's project window.
- Choose **Restore** from the **Module** menu.
  The Restore *Module* dialog will open.



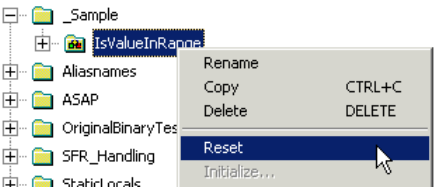
- Select one **TMB** file and click **OK**.
  Tessy will insert the module.

Reset a Module

Resetting a module will delete the complete test data (test cases) of all test objects. Furthermore the passing directions of the interface of all test objects will be set to default values.

To reset a Module, do the following:

- Select the module and choose **Reset** from the context menu.

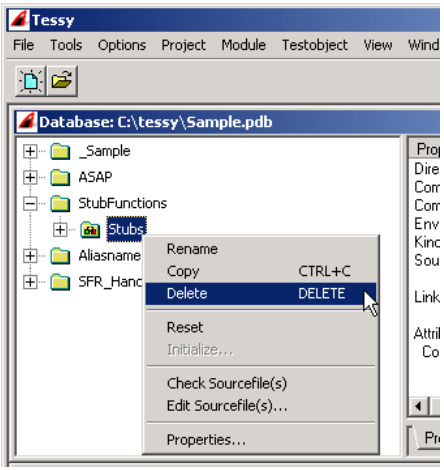


Delete a Module

To delete a module, do the following:

- Select a module you want to delete.
- Choose **Delete** from the context menu or use `[Del]` on the keyboard.

  Tessy will remove the module. This will also delete the module folder from the hard disk.



Specifying the Test Environment

The test environment of a module specifies the common settings to execute a test run, such as

- Target Compiler :: Debugger/Emulator combination
- Additional Compiler and Linker options
- Source files

With exception of the used source files, you have to enable and adjusting common settings of a specific test environment (e.g., Cosmic HCS12 :: HiTOP) by using the Tessy Environment Editor (s. Using Default Configurations).

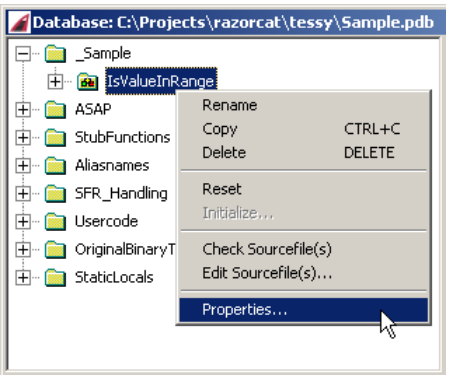
As soon as your target environment has been enabled, you can select it within the environment list box of the General Tab tab (page 99) of the Module Properties dialog (s. below).

Module Properties

The *Module Properties* dialog is used to determine the test environment and additional settings for your testing.

To specify a test environment, do the following:

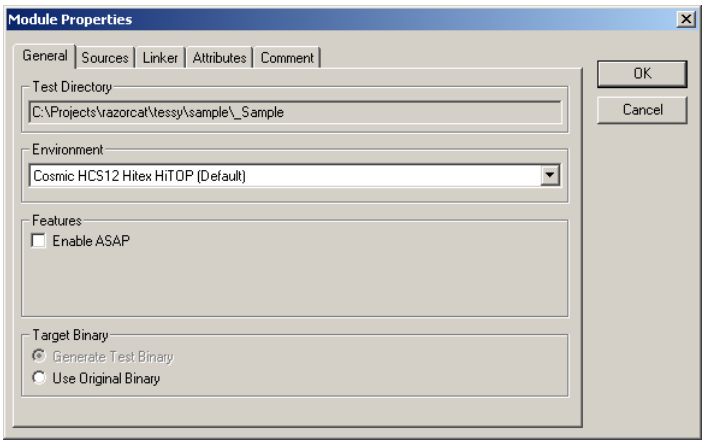
- Select a module and choose **Properties…** from the context menu.



The Module Properties dialog will open.

General Tab

The *General Tab* is used to determine the test environment. Following options are available:

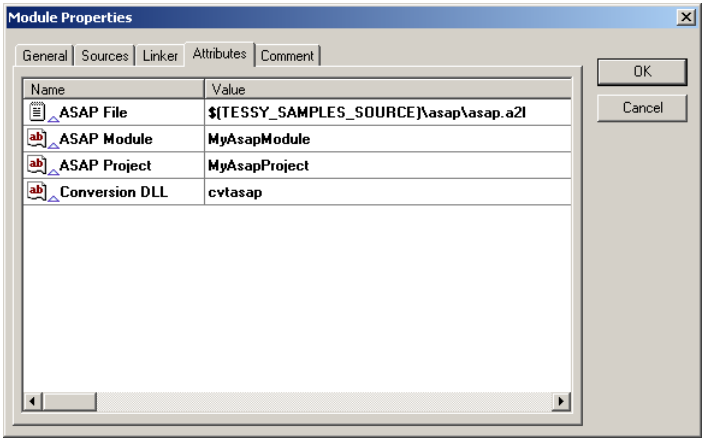


- ## Test Directory
  The path has been specified during database creation and is not adjustable here.
- ## Environment
  Specifies your [target compiler :: debugger/emulator/simulator] combination to be used for the test execution. You have to enable your test environment by using the **Tessy Environment Editor** (s. Using Default Configurations).
- ## Target Binary
  Generate Test Binary: The test binary will be generated and compiled by Tessy (default).
  Use Original Binary: You can use your original binary build from your own development environment. If you choose *Use Original Binary*, the **Linker** tab will disappear and a new tab **Binary** will be shown (see Linker / Binary Tab).

***Note**: The binary test is currently only available for HiTOP and TRACE32 emulators. Please refer to the application notes "Original Binary Test" from the* **Help|Documents** *menu for more information on this topic.*

- ## Features
  Enable ASAP: Tessy provides a close integration to the ASAP standard, allowing the usage of ASAP conversion rules for physical to integer conversion of test data. Please refer to the application notes "Using ASAP Information" from the Help|Documents menu for more information on this topic.

If ASAP is ticked, you will find additional attributes within the **Attributes** tab in which you have to specify your ASAP file.



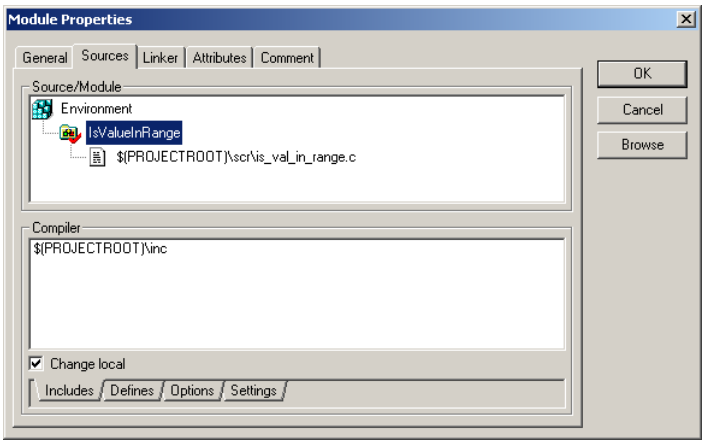
Sources Tab

On this tab you can add the source files to be tested. All exported functions will be displayed if the module is opened.

By default, Tessy displays the module folder in the **Source/Module** pane in which the source files are added.

Some additional compiler options may be specified on module level by selecting the module entry, other options may be specified for each source file in the list. Such local changes displays Tessy with a red tick in front of the module or source file name. The check-box “Change local” will be ticked.

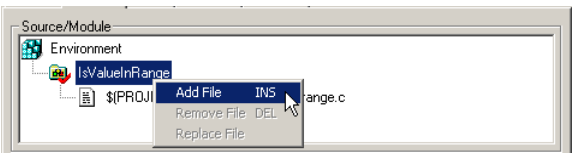
Some of the displayed options (e.g., include paths) in the lower Compiler pane may be specified in the Environment Editor (s. Using Default Configurations) and will be inherited from there.



Add Source Files

To add source files:

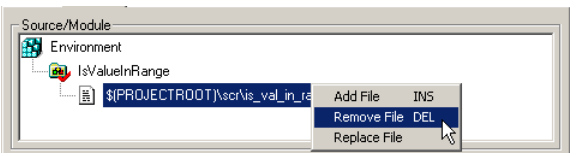
- Select either the module folder, a source file or right-click into the Source/Module pane (you may also use the **Browse** button). Choose Add File from the context menu:



Replace or Remove Source Files

To remove a source file:

- Select a source file and choose **Remove File** from the context menu or use `[Del]` from the Keyboard.



To replace a source file:

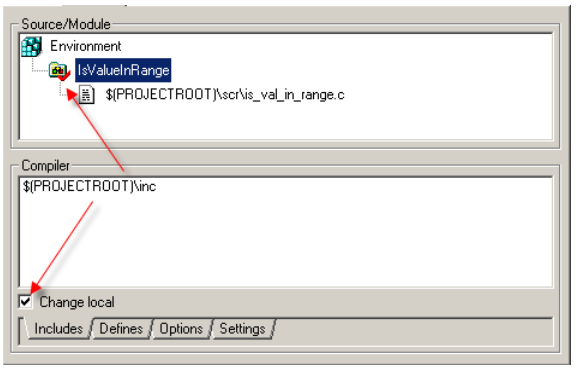
- Select a source file and choose **Replace File** from the context.
- From the next dialog, select another source file.

Using the Lower Compiler Pane

The lower **Compiler** pane displays information about the item selected from the upper **Source/Module** pane. What kind of information depends on the tab {Includes|Defines|Options|Settings} you select. You may specify additional settings in each of the provided tab.

Module options apply to all source files unless otherwise specified on file level. File options apply to one selected source file and may replace options specified on module level.

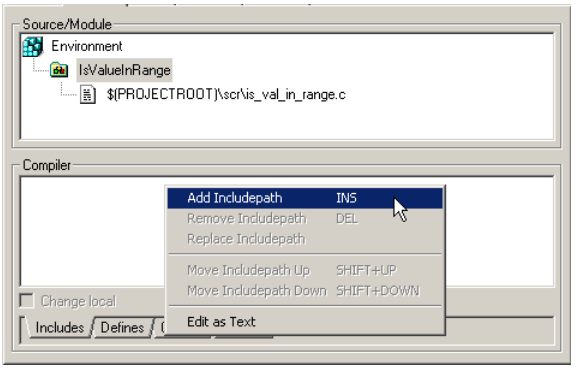
Such local changes displays Tessy with a red tick in front of the module or source file name. The check-box *Change local* will be ticked.



Includes Tab

Within the **Includes** tab, you may specify “include paths”.

- Right-click into the Includes pane and choose **Add Includepath** form the context menu.



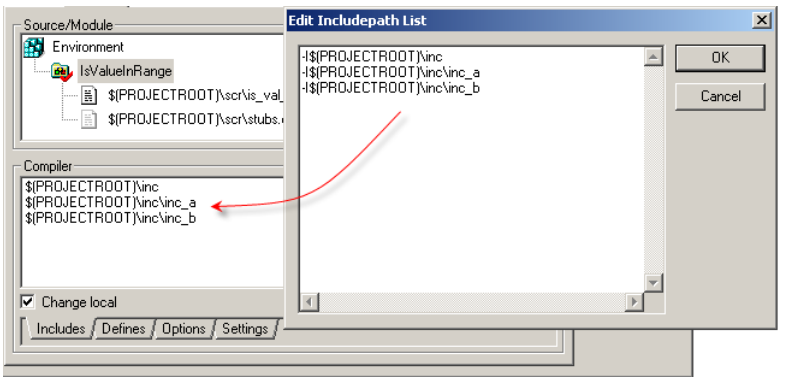
- From the *Browse For Folder* dialog, choose an appropriate path.

If possible, Tessy will always use relative path names instead of absolute path names, e.g. $(PROJECTROOT)\inc (s. The Project Root on page 79).

Edit as Text

You may also add “include paths” with option **Edit as Text**. This is useful if you have a configuration file which defines the “include paths” for your source code.

- Right-click on the Includes pane and choose **Edit as Text** from the context menu.
- Copy-n-Paste your include path list into the edit field of the dialog.
- Click OK. *The paths will be inserted.*

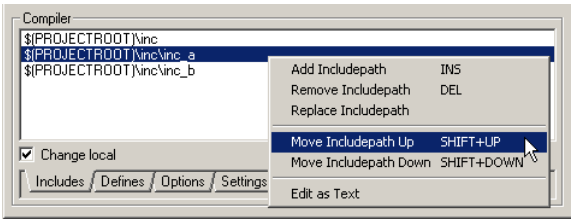


***Note**: Please use the appropriate compiler option for your target compiler (e.g. gcc: –I) to specify include paths in the list.*

Move Includepath

If necessary, you can change the row sequence of the include path list.

- Right-click one of the inserted include paths and choose **Move Includepath Up** or **Down** from the context menu. You may also use `[Shift]+[Up]` or `[Shift]+[Down]` keys on a selected item.



Replace or Remove Include Paths

To remove a include path:

- Select a include path and choose **Remove Includepath** from the context menu or use `[Del]` from the Keyboard.

To replace a include path:

- Select a include path and choose **Replace Includepath** from the context.
- From the next *Browse For Folder* dialog, select another include path.

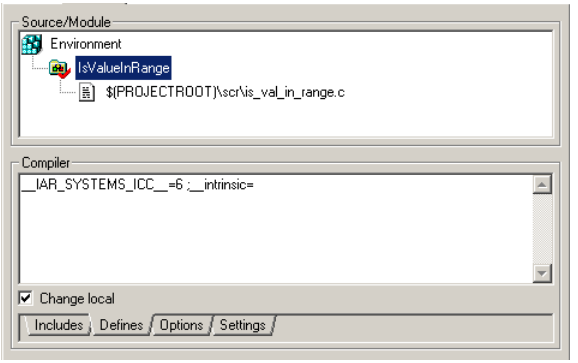
Defines Tab

Within the **Defines** tab, you may specify preprocessor defines, if necessary.

***Important note**: Please enter all definitions without the normally used option of your target compiler, e.g. –D. Every define has to be separate by a comma or semicolon.*

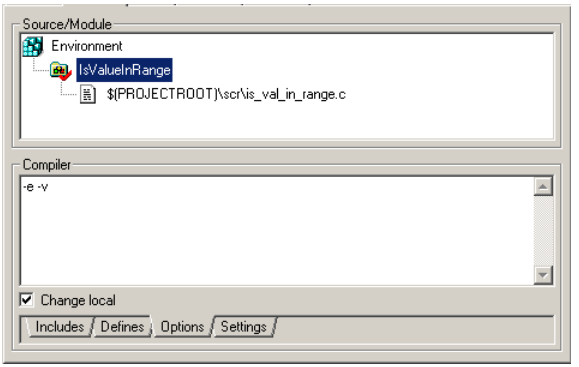
Example:

```
__IAR_SYSTEMS_ICC__=6 , __intrinsic=
```



Options Tab

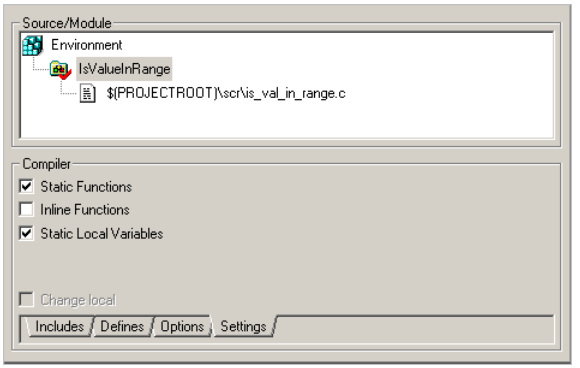
Within the **Options** tab, you may specify additional directives for your target compiler. Please mind, that preprocessor directives and include paths has to be specified within the **Defines** tab respectively **Includes** tab.



Settings Tab: module selected

Within the **Settings** tab you can force Tessy to enable static functions, static local variables or hide inline functions.

These options apply to the whole module though it may be selected only when the module entry is selected.

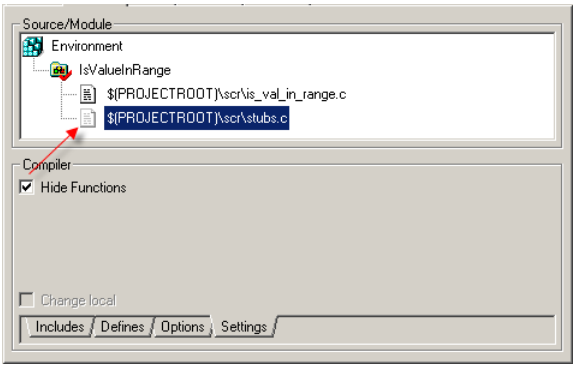


- **Enable Static Functions**
  If this box is not checked, no static function will be listed in the test object list of the module.
- **Enable Inline Functions**
  If this box is not checked, no function with the keyword "inline" will be listed in the test object list of the module.
- **Enable Static Local Variables**
  Static local variables within functions may be used as normal input or output variables within Tessy. The interface parser will recognize those variables and they may be set and evaluated.

***Note**: Static local variables require special handling. Please refer to Input Values for static local Variables for more information.*

Settings Tab: source file selected

Within the **Settings** tab you can force Tessy to hide functions of certain source files from being displayed in the list of test objects.



- **Hide Functions**
  This option may only be applied on source file level and though requires a source file to be selected. Checking this option box will gray out the icon of the selected source file indicating a hidden source file.

  This option is useful for additional source files needed for testing (e.g. implementation of stub functions for called functions). Functions from those helper files may be hidden from the list of test objects since they are not relevant for testing and reporting.

Symbol Appearance in the Source/Module Pane

The icon in front of a module name and every source file name indicates the current status and option settings for this item. Local changes, such as additional compiler directives, include paths, etc. indicate Tessy with a red tick in front of the module or source file name. The check-box “Change local” will be checked in the respective tab.

Icon	Description
	No special module options defined.
	Module options defined.

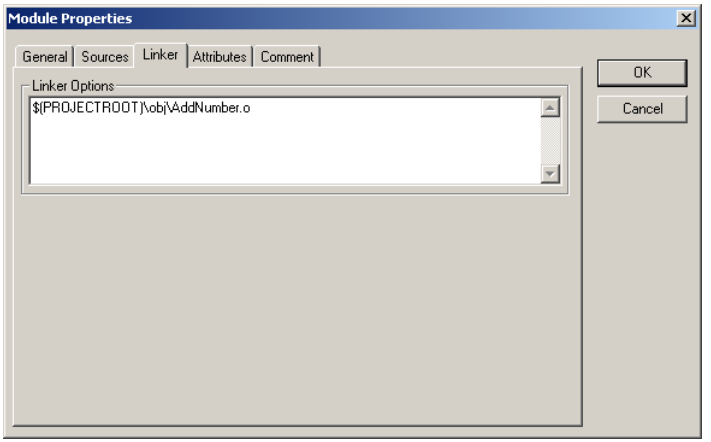
Icon	Description
	No special source file options defined.
	Source file specific options defined.
	Source file is hidden. No special options defined.
	Source file is hidden. Source file specific options defined.
	Newly added file.
	Newly added file. Source file specific options defined.
	Newly added file. Source file is hidden.
	Newly added file. Source file is hidden. Source file specific options defined.

Linker / Binary Tab

Depending on the setting of *Test Binary* in the General Tab, either the **Linker** or the **Binary** tab will be shown.

Linker tab (setting: generate test binary)

You may specify libraries, object files or other linker command line options within the **Linker Options** pane. All settings are used to build the test driver executable.

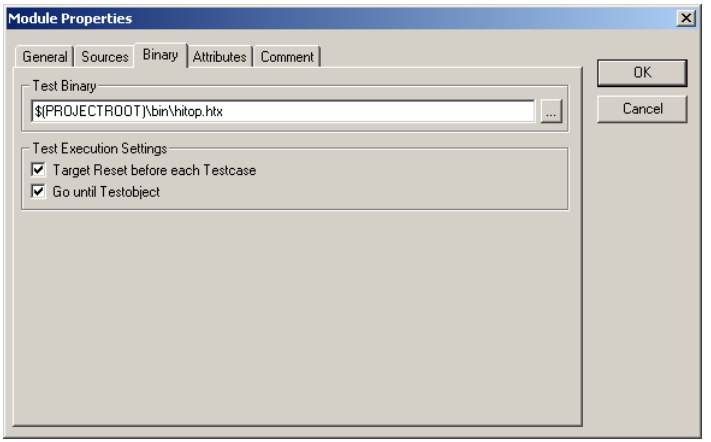


Binary tab (setting: use original binary)

In the **Binary** tab, you need to enter the executable binary file to be used for test execution on the target.

In case of HiTOP , select the *.htx file in the respective build directory of your development environment. The file name extension required for the *Test Binary* depends on the target environment to be used.

Please refer to the application note "Original Binary Test" from the **Help|Dokuments** menu for details on this test method and the required settings for individual target system configurations.



In the *Test Execution Settings* section, you may specify following settings:

- **Target Reset before each Testcase**
  Resetting the target system before executing each *test case* is the default behavior.

  This ensures, that each *test step* will be started with an initial state of the system.

- **Go until Testobject**
  Running the whole binary application until the beginning of the *test object* for each *test step*.

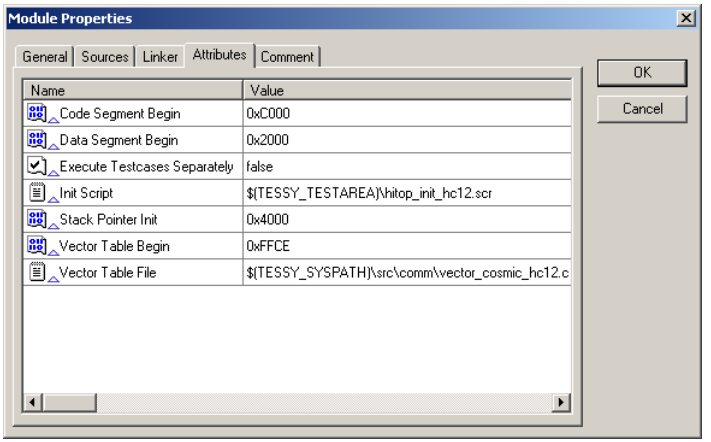
  This requires, that the *test object* will be called at least once. If you don't select this option, you will need to use the *user code* scripting facilities to run until the *test object*.

Attributes Tab

The **Attributes** tab specifies settings required by the compiler or the target environment of the module.

Most attributes were presets. They have been specified in the Tessy Environment Editor for a given target environment (s. Using Default Configurations).

Attributes are only displayed if they have gotten the property “visible” in the Environment Editor. The Attribute pane may be empty.



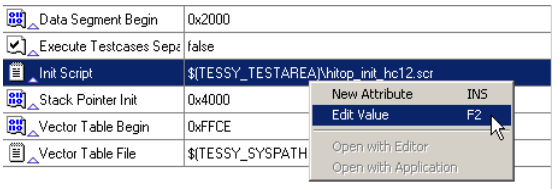
You may change the default values or add new attributes to the Attributes pane. Changes are carried out only locally and don't influence other modules. This meant that default values are preserved.

***Note**: You must use the Environment Editor if you want to use added attributes also in other modules for the given target.*

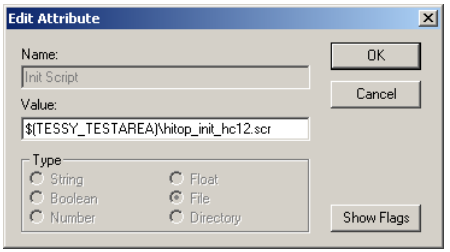
Edit Attribute Values

There are some possibilities to edit an attribute value. This is also dependent by the used data type:

- Using the context menu on a selected item.
- Using short cuts on a selected item, e.g. [F2]
  The inline editor will open.



- Double-clicking the attribute name.
  The Edit Attribute dialog will open.



The attribute will be displayed in bold face if the default value has changed.

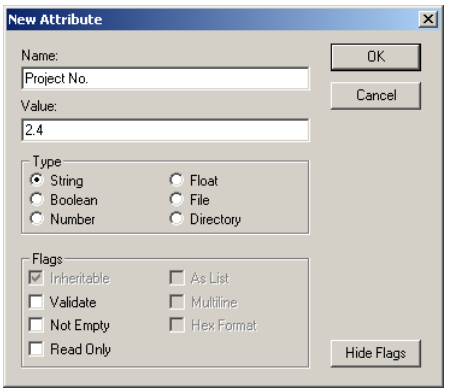
Code Segment Begin	0xC000
Data Segment Begin	0x2000
Init Script	**$(TESSY_TESTAREA)\hitop_my_init_hc12.scr**
Stack Pointer Init	0x4000

Insert Attributes

You may add new attributes to the Attributes pane. This new attribute will only be available for the given module and don't influence other modules. There are different attribute types available: `String, Boolean, Number, Float, File` and `Directory`.

To create a new attribute:

- Right-click in the Attributes pane or an already inserted attribute and choose **New Attribute** from the context menu. *The New Attribute dialog will open.*



- Enter an attribute name and choose an appropriate **Type**, e.g. *String*.

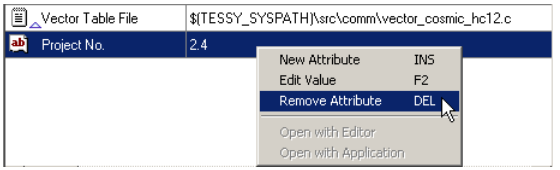
  Additional, every Type may have specific attribute flags. This depends on the Type used.

Remove/Reset Attributes

Only user-defined attributes can be removed from the attribute pane. You cannot remove default attributes. In that case, you may only **Reset** the value to his default state, if changed.

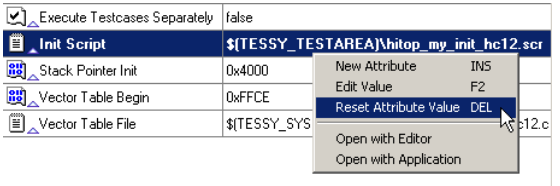
To remove a user-defined attribute:

- Right-click an attribute and choose **Remove Attribute** from the context menu, or use the `[Del]` key on a selected attribute. *The attribute will be removed.*



To reset a default attribute:

- Right-click an attribute and choose **Reset Attribute** from the context menu, or use the `[Del]` key on a selected attribute. *The value will be reset.*



Comment Tab

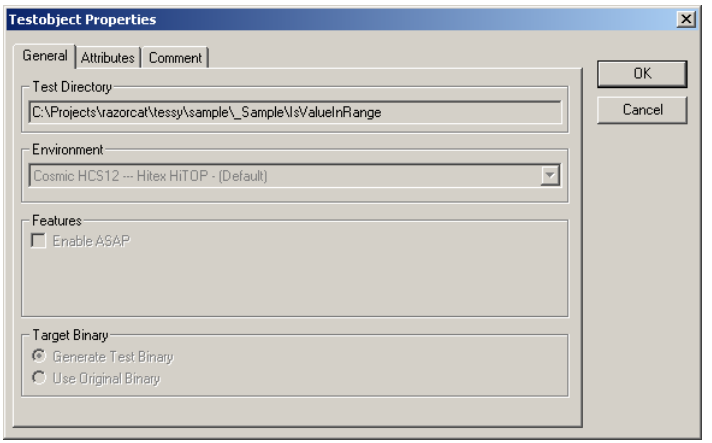
This property provides an editable textbox to be used for comments by the tester.

Test Object Properties

As for a module, there is also a properties dialog available for a test object. You can review or change settings, but most of them are already done within the module properties dialog or the Environment Editor.

To open the test object properties dialog:

- Select a test object from the module folder and choose **Properties** from the context menu. *The Test Object Properties Dialog will open.*

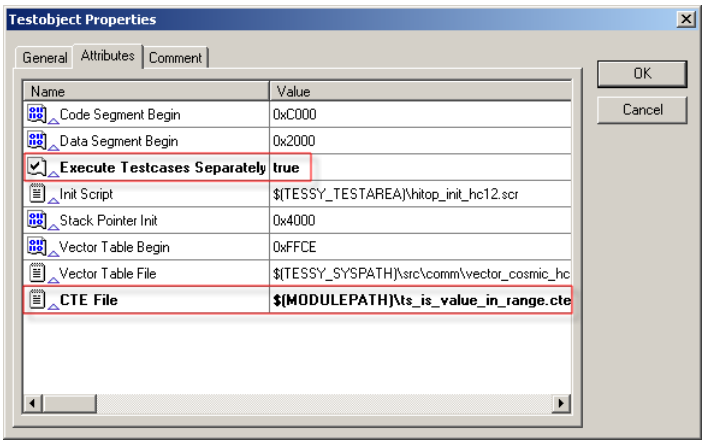


General Tab

This tab is read only and displays all settings done within the module properties dialog.

Attributes

This tab displays the module attribute settings by default (if any).



There are additional test object specific attributes available:

- **Execute Testcases Separately**
  Normally, this attribute is not visible (s. Environment Editor). Execute Testcases separately will be set on true, if you have checked this option within the Execute Test dialog. This attribute is taken into consideration for every test object individually when you carry out batch tests (s. Batch Test). You may set this attribute within in TEE on `true` if you want to use this option by default for all test objects.
- **CTE File**
  This attribute becomes visible if test cases have been created and exported to Tessy by using the CTE. After exporting of test cases, Tessy will also copy the CTE document into the respective module folder and rename then the document in ts_*the_testobject_name*.cte. The CTE document **must** be stored in the module folder!

All attributes will be displayed in bold face if the default value has changed.

***Note**: You may change every setting for each test object locally. We do not recommend to change compiler or target specific settings for individual test objects. Those settings should only be applied for the entire module using the module properties dialog or the Environment Editor.*

Comment

This property provides an editable textbox to be used for comments by the tester.

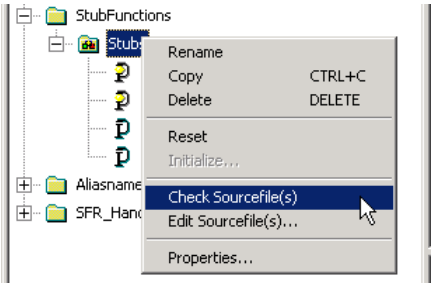
Common Module Options

From the module context menu you have additional options for source files of the module available. In case of problems while opening the module you can check your source files using the target compiler. Errors may be corrected directly in your preferred editor.

Check Source Files

You can check your source files for errors as follows:

- Select a module and choose **Check Sourcefile(s)** from the context menu.



Tessy will now use the respective target compiler to check all source files of the module. In case of errors, Tessy lists all error messages generated by the target compiler within the message window.

Note: *This checking is highly recommended in case of any problems while opening the module. The target compiler will print out more precise error messages than the Tessy analyzer.*

Edit Source Files

The menu item **Edit Sourcefile(s)** allows you to edit the source files of a module in your preferred editor.

You may specify the editor to be used in the **tessy.conf** file as follows:

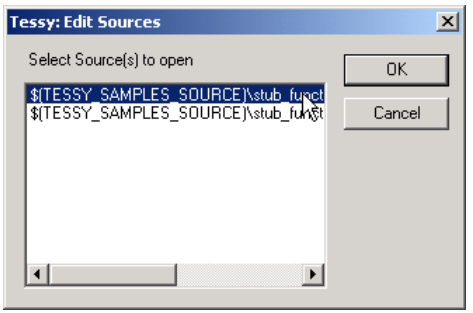
- Open the configuration file `tessy.conf` with an editor `(C:\Documents and Settings\[Login Name]\Application Data\Razorcat\Tessy\[Version]\config\).`
- Enter your preferred editor within the `[Options]` section, e.g. `Editor=notepad`

Note: *You may access the tessy.conf directly within from Tessy by using* ***Options|Edit Settings...***

To edit your source file(s), do the following:

- Select a module and choose **Edit Sourcefile(s)** from the context menu.

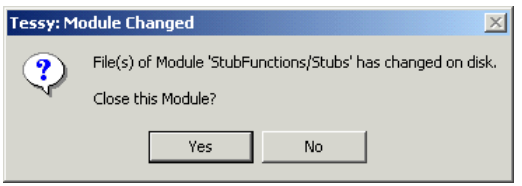
  If you have more than one source files, following dialog will open.



- Choose the respective file(s) from the dialog and click **OK**.

Source Files Changed

If the module folder is open while saving a changed source file, the following message will appear:



Tessy will close the module folder **Stubs** of the project **StubFunctions** to ensure that a changed source file will be analyzed again.

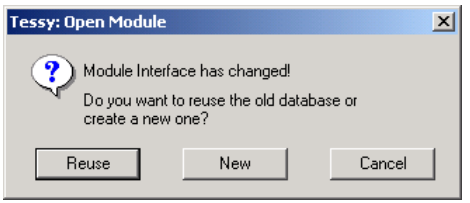
You may do the following:

- Choose **Yes.** Tessy will close the module.
- Choose **No** to ignore the message. The module will stay open.

***Important note**: Remember to generate and compile the test driver, since the implementation of the test object may have changed.*

Changed Interface

If you open a module and the interface of a test object has changed, e.g. after editing of a source file, the following dialog may appear:



***Note:** Source files are often modified extensively during the later stages of development, thus making it necessary for a new test (e.g. a regression test) to take place. This can alter the interface of an existing test object. Tessy enabled you to **Reuse** test data that has already been entered.*

You may do the following:

- Click **Reuse** in order to start the IDA (Interface Data Assigner) for reusing your test data (see Interface Data Assign Editor)
- Click **New** if you want to **delete** all test data and restart from scratch.

Interface Analysis

If you open the module for the first time, source files and the respective header files will be scanned and a list of dependencies will be generated. This allows to check for file changes

when opening the module a second time before analyzing the interface. If no file changes occurred, no comparison of the IDB will be carried out, thus saving time when opening modules.

While opening the module, Tessy will create a special folder **.idbbuild** in the respective module folder to store all necessary information.

Advanced interface analysis is available by default and should be switched off only in case of problems. You have to change following option by using the **tessy.conf** in following path:

%APPDATA%\Razorcat\Tessy\[Version]\config

UseIDBMake=0 [off]

UseIDBMake=1 [on]

Note: *You may access the tessy.conf directly within from Tessy by using* ***Options|Edit Settings...***

Updating .idbbuild folder

In some cases you have to update the **.idbbuild** folder. For instance, this will be necessary if you have changed your **$(PROJECTROOT)** or some environment variables have been changed (if you have simply copied the whole project folders into another path).

You may also update the **.idbbuild** folder if you receive an error message from "**idb.mak**" while you open the module. If you do not update the **.idbbuild** folder, the module could not open.

To update the **.idbbuild** folder:

- Hold briefly the [Ctrl] key pressed while you double-click the module or click .

Tessy Environment Editor TEE

Configuration Management

Tessy provides the so called Tessy Environment Editor **TEE** to configure compiler and target environments. With the installation of Tessy, the configurations for all supported compiler and target environments (including necessary settings and files) were copied to the Tessy installation directory. You need to enable the compiler and targets that you want to use with Tessy. The default settings may need to be adapted to your needs (i.e. the installation path of the compiler or target debugger is one of the settings that normally need to be changed to your local values).

Settings for compiler and target environments which have already been used with a previous version of Tessy were also taken over during installation.

The TEE configuration management allows you to create variants of compiler and target settings and assign them to a module. You may also save your settings in a specific configuration file to use this file with a given project database. This allows easy sharing of specific environment configurations between developers of the same development team.

As a result you have all your basic settings at one central place (e.g. include paths, additional compiler options, etc.). Once configured, you may start testing immediately using the required configuration for all your modules.

Important note: TEE provides only predefined configurations for all supported compiler and target environments. We recommend checking these settings for your specific environment, e.g. linker files, project files, makefile templates!

Starting the TEE

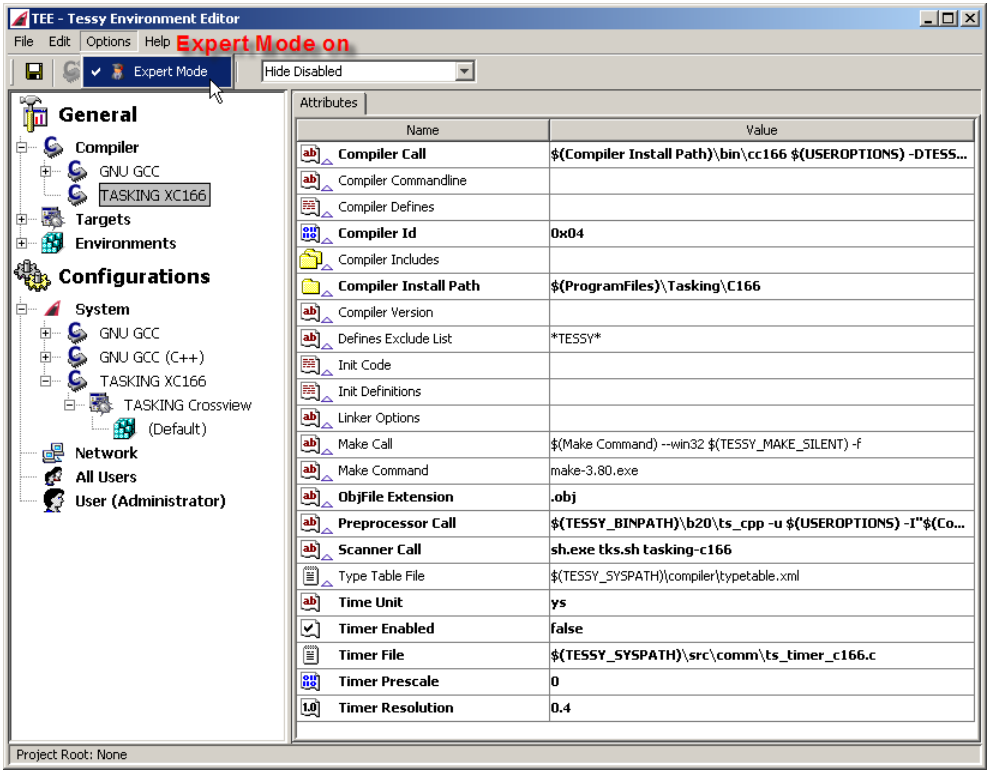
We recommend starting **TEE** from within Tessy by using the **Options|Edit Environment…** menu if a project database is open. This ensures that **TEE** also uses the so called **PROJECTROOT** from the project database if specified (s. Assign a Project Root).

TEE will then start either with a specific custom configuration file (s. "Create a New Custom Configuration File") or with a default configuration provided by TEE automatically.

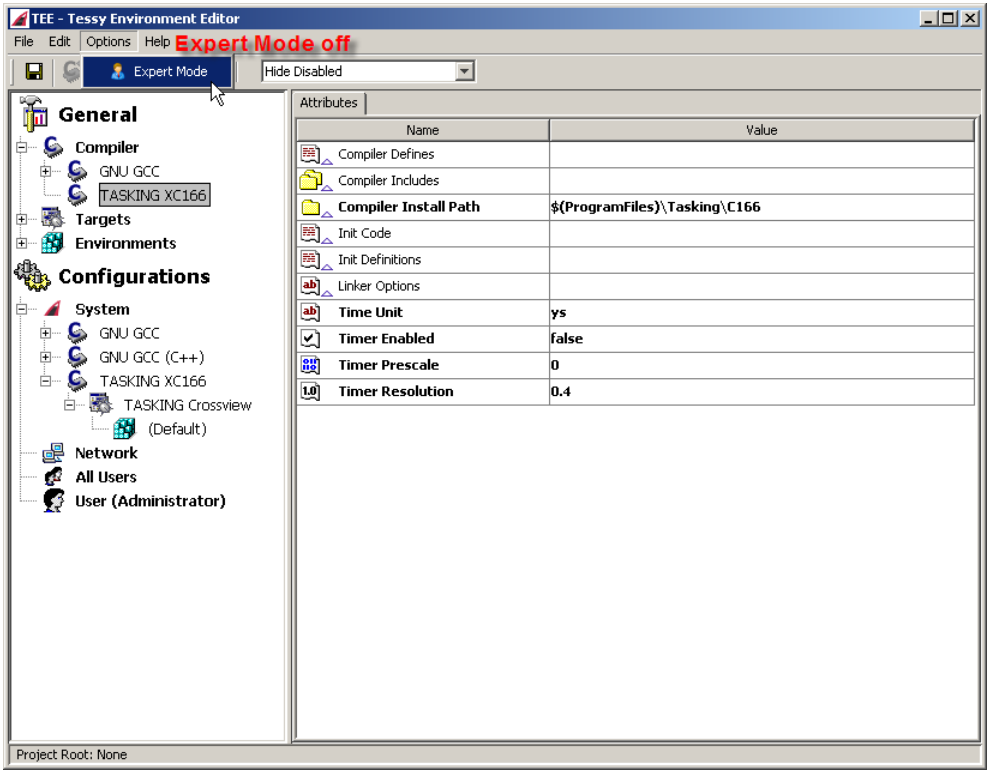
You may also start TEE from the Windows Start menu **Programs|Tessy 2.6|Tessy Environment Editor.** The Environment Editor will then start with a default configuration.

TEE Expert Mode

TEE provides a so called **Expert Mode** which displays all attributes settings in the right hand pane.



When the expert mode is not activated, only basic attributes are visible for the user. The later case is the default setting and appropriate if your test environment has been already configured for the end user.



TEE Structure

There are two panes in the environment editor **TEE**:

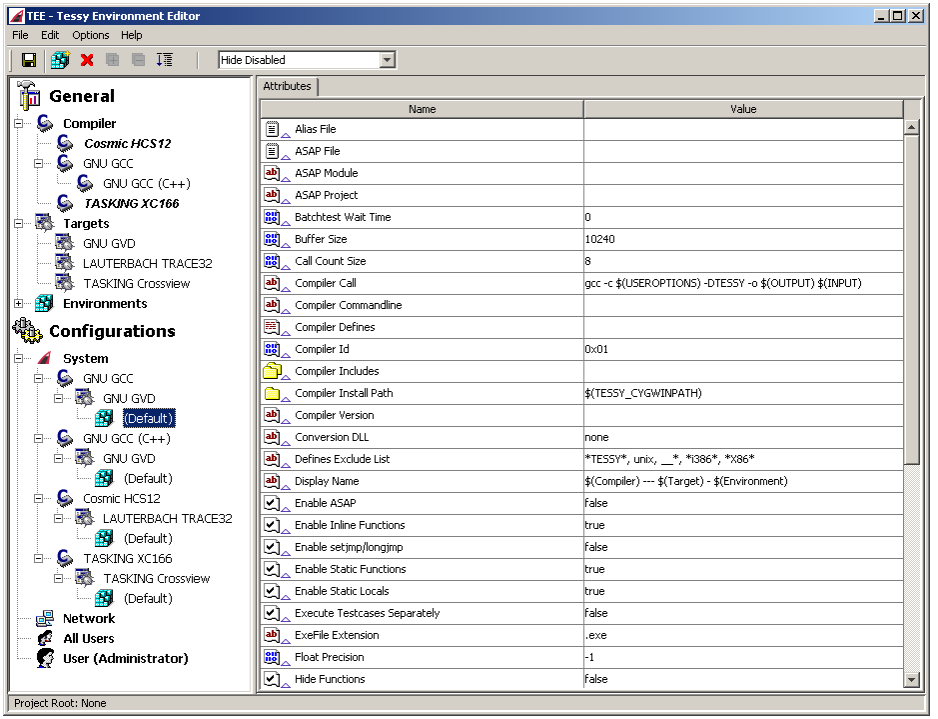
Left-hand pane

The left-hand pane provides two sections, **General** and **Configurations**.

- **General:** lists all supported **Compiler** and **Target** environments which were configured with default settings.
- **Configurations:** lists all supported compiler and target combinations. The settings of these combinations have been inherited by the **General** section (s. Using Default Configurations below).

Right-hand pane

The right-hand **Attribute** pane displays all attributes of an item as soon as it was selected in the left pane.



***Note**: You can refresh all the panes in TEE by pressing `[F5]`.*

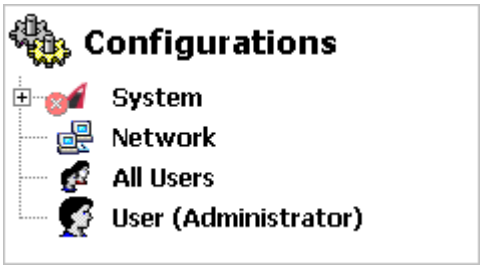
TEE and User Access Rights

Depending on the access rights of your user account, you may be able to change parts of the default configurations or not. Normally, you will need administrator privileges to change Tessy's default configurations (factory settings) within section **General** and **Configurations** (except **User** subsection).

If you don't have enough privileges (if you are a restricted user), TEE will inform you and save your changes in a user specific file (s. "TEE Configuration Files" below).

TEE Configuration Files

All displayed settings will be stored in a set of configuration files with extension "`*.xml`". Every configuration group (**System, Network, All Users** and **User**) of the **Configuration** section will have (a) separate configuration file(s).



The configuration file of **System** contains the default settings for all supported compiler and target environments and has been installed with Tessy. As soon as the **TEE** is launched, a configuration file will be generated for the respective **User**. The configuration files for **User** respectively **All Users** will be empty, as long as no configurations were created (e.g. copied into this subsection).

Unlike the configuration file of **System,** the configuration file for **User** respectively **All Users** contain only compiler/target configurations which you have copied into these

subsections. TEE will always merge the configuration files during startup to display the resulting settings for the respective configuration groups.

The configuration files for **System, Network, All Users** and **User** will be stored in following default folders:

[System]

`C:\Program Files\Razorcat\Tessy_2.6\config`

- **`configuration.xml`**
  Contains settings for the **General** and **Configurations** section.
- **`configuration_default.xml`**
  Represents the factory settings of Tessy. During installation of new Tessy versions, the settings within this file will be used to update `configuration.xml` if necessary.
- **`network_configuration.xml`**
  s. Network below.

[Network]

`C:\Program Files\Razorcat\Tessy_2.6\config`

- **`network_configuration.xml`**
  Specifies a `configuration.xml` which was stored in your network. The syntax is as follows:

  <?xml version="1.0" encoding="ISO-8859-1"?>
  <link target="[PATH]\\configuration.xml"/>

[All Users]

`%ALLUSERSPROFILE%\Razorcat\Tessy\[Version]\config`

- **`configuration.xml`**
  Contains settings for this section (if any).

[User]

`%APPDATA%\Razorcat\Tessy\[Version]\config\`

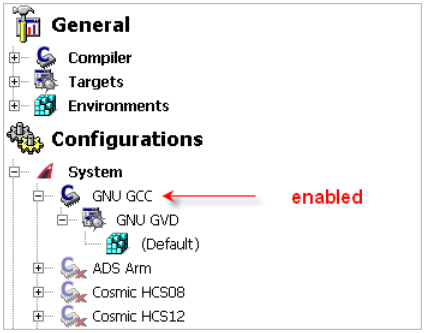
- **`configuration.xml`**
  Contains settings for this section (if any).

- **`configuration_system.xml`**
  Contains all your changes made in the General and Configurations sections if you don't have enough privileges (restricted user) to change the system configuration file of the Tessy installation. Another User account on the same computer will not be updated.

Using Default Configurations

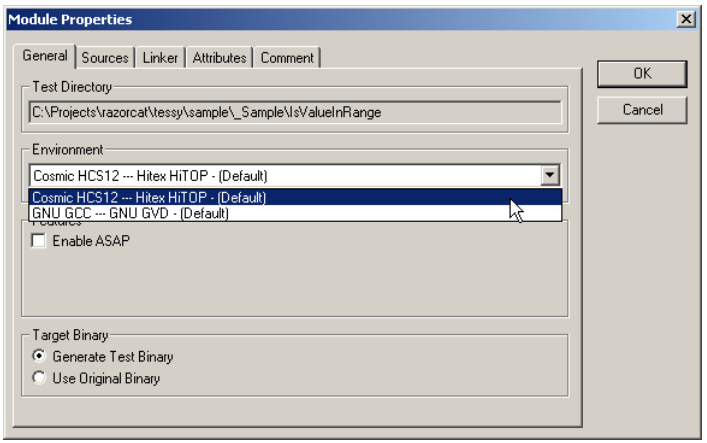
TEE provides predefined configurations for all supported compiler and target environments. You will find these configurations in the **Configuration** section under **System**. The settings of these combinations are inherited from the **General** section.

By default, at least the [**`GNU GCC::GNU GVD::(Default)`**] Environment will be enabled (and all your compiler/target environments used in previously installed versions of Tessy, if any).



All other predefined configurations are disabled by default. TEE will mark the respective icon with a red cross to indicate, that a configuration is disabled (s. Symbol Appearance).

A configuration will be available within the **Environment** list box in the **General** tab of the **Module Properties** dialog (within the Tessy main window; s. Module Properties) as soon as an appropriate Compiler/Target combination has been enabled (s. Enable Configurations).



Enable Configurations

A configuration consists of a combination of **Compiler**, **Target** and a **Default** environment.

You have to either enable both a **Compiler** and a **Target** in the **General** section or you may enable the desired configuration within the **System** section directly (s. below).

General Section: Enable Compiler and Targets

Example:

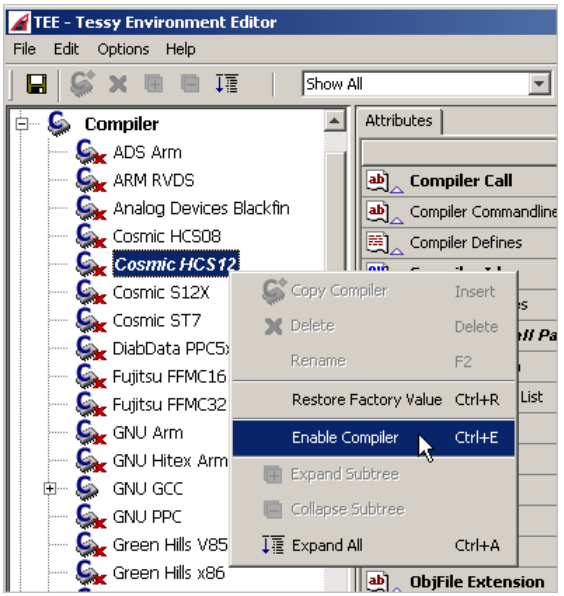
Compiler :: `Cosmic HCS12`
Target :: `Hitex HiTOP`

Note: *Choose* ***Show All*** *from the Filter box to display all supported compiler and target environments.*

Enable Compiler:

- In the left-hand pane, open the **Compiler** tree in the **General** section and select, e.g. `Cosmic HCS12`.

- Choose **Enable Compiler** from the context menu or use `[Ctrl]+[E]`.

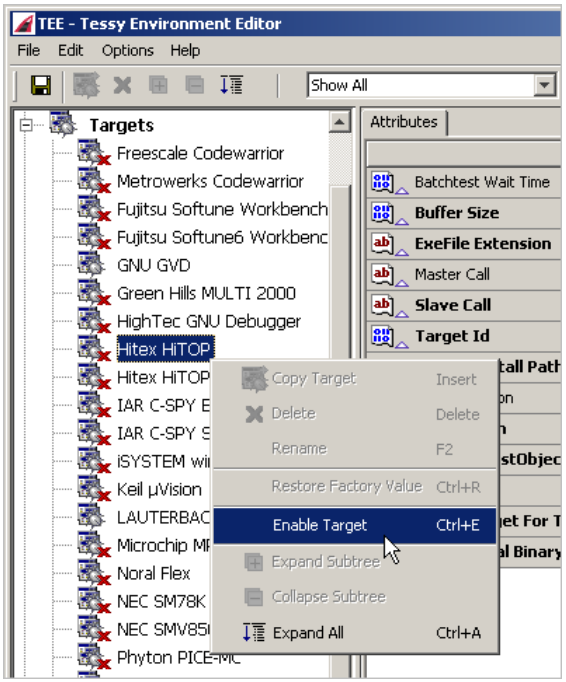


This will enable your compiler.

TEE will remove the red cross from the icon to indicate, that this compiler is enabled (s. Symbol Appearance).

Enable Targets:

- In the left-hand pane, open the **Targets** tree in the **General** section and select, e.g. **`Hitex HiTOP`**.
- Choose **Enable Target** from the context menu or use `[Ctrl]+[E]`.



This will enable your target.

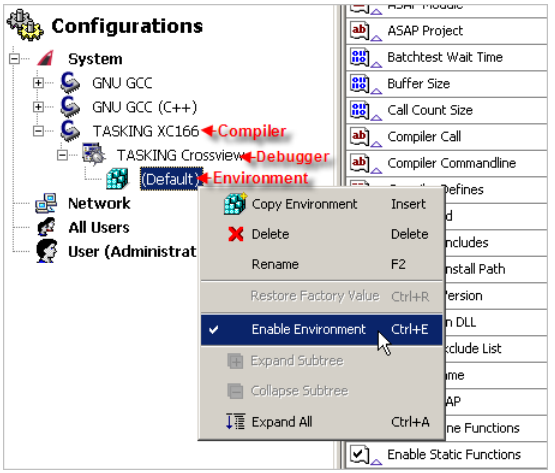
TEE will remove the red cross from the icon to indicate, that this target is enabled (s. Symbol Appearance).

System Section: Enable Configurations

You can only use supported compiler - target combinations in TEE. Therefore you may activate your preferred environments directly in the **Configurations|System** section. The **System** tree contains all supported compiler - target combinations, e.g.

```
:: (Compiler) Tasking XC166
:: (Debugger)Tasking Crossview
:: (Environment) (Default)
```

- To enable a configuration, select the **(Default)** environment and choose **Enable Environment** from the context menu.

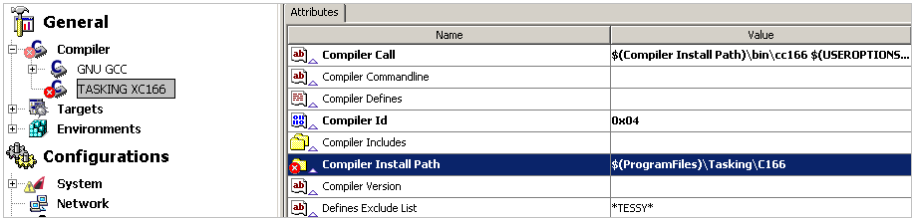


Customizing of Enabled Configurations

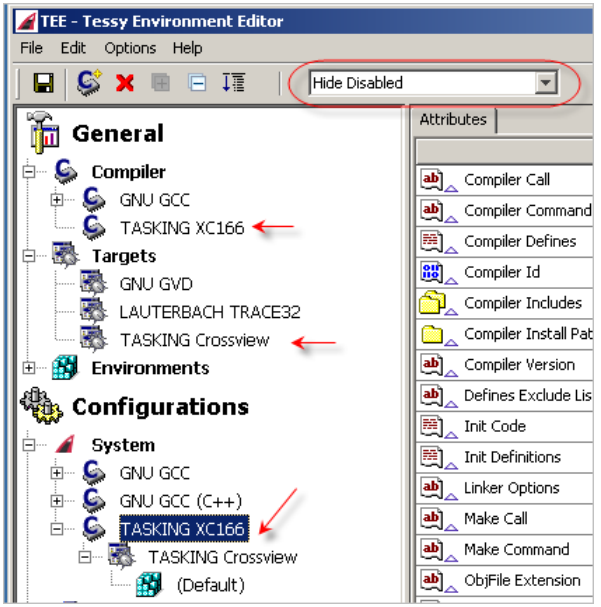
TEE provides only predefined configurations for all supported compiler and target environments.

Normally, you need to change some settings for your specific environment. Some of the settings will be checked for validity.

TEE will check all enabled configurations and displays error and warning signs as soon as an error has been found, e.g. the **`Compiler Install Path`** must be corrected.



As soon as a **Compiler** and an appropriate **Target** are activated, the respective configuration will also be available in the **Configurations** section under **System**.



***Note**: Choose **Hide Disabled** configurations from the filter box to hide all configurations that have not been enabled. This may give you a better overview about already enabled configurations.*

Advanced Configurations

TEE allows you to derive variants of compiler and target settings and assign them to a module. This enables you to create specific configurations for different projects. For instance, you may use a specific make template file, linker file or specific libraries and compiler include paths.

There are two ways to perform such advanced configurations.

- You may copy already created configurations and change their settings to suit your needs (s. Copying Configurations).
- You may also save your settings in a specific configuration file to use this file with a given project database. This allows easy sharing of specific settings between developers (s. Creating Configuration Files).

This is the recommended method to use configurations.

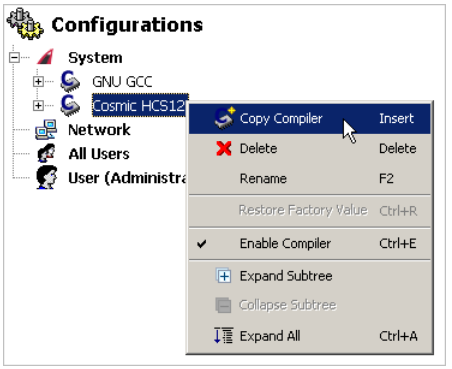
Note: *Since Tessy 2.6 the cloning feature has been removed from TEE.*

Copying Configurations

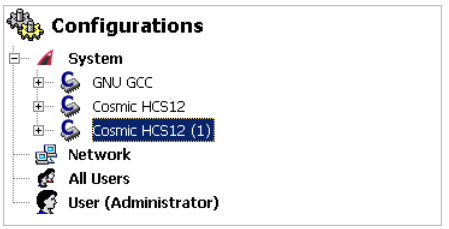
This feature is available in the **Configurations** section of TEE and allows copying an already existing configuration. This is the recommended method to create your own configurations, because all required default settings will also be copied.

To copy a configuration:

- Within the **System** section, select one configuration, e.g. **`Cosmic HCS12`**
- Choose **Copy Compiler** from the context menu. Alternatively, drag-n-drop the configuration onto the **System** section icon or one of the other configuration groups **Network, All Users** or **User**.

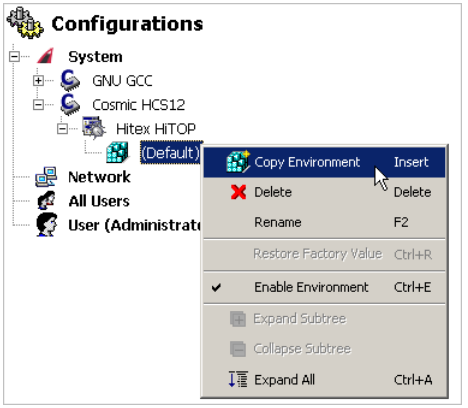


TEE will insert an additional entry **`Cosmic HCS12 (1)`**

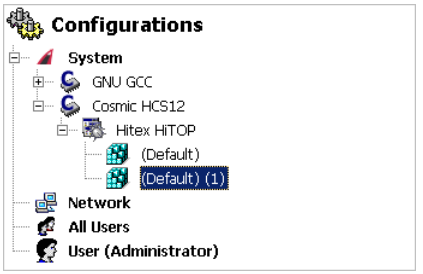


- You may rename this item to describe your environment.

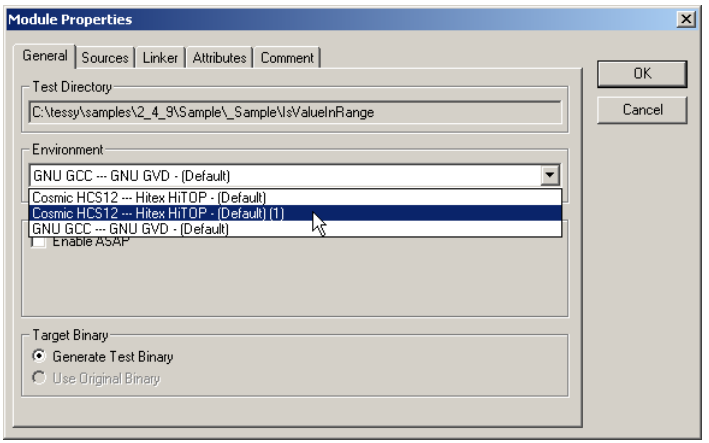
It is not always necessary to copy the whole configuration. You may copy only the target entry or the environment entry to create a different configuration for a given compiler.



TEE will insert an additional entry ***(Default) (1)***



This additional configuration will be available within the **Environment** list box in the **General** tab of the Module Properties dialog.


Module Properties
General
Sources
Linker
Attributes
Comment
OK
Cancel
Test Directory
C:\tessy\samples\2_4_9\Sample_Sample\IsValueInRange
Environment
GNU GCC --- GNU GVD - (Default)
Cosmic HCS12 --- Hitex HiTOP - (Default)
Cosmic HCS12 --- Hitex HiTOP - (Default) (1)
GNU GCC --- GNU GVD - (Default)
Enable ASAP
Target Binary
Generate Test Binary
Use Original Binary

Creating Configuration Files

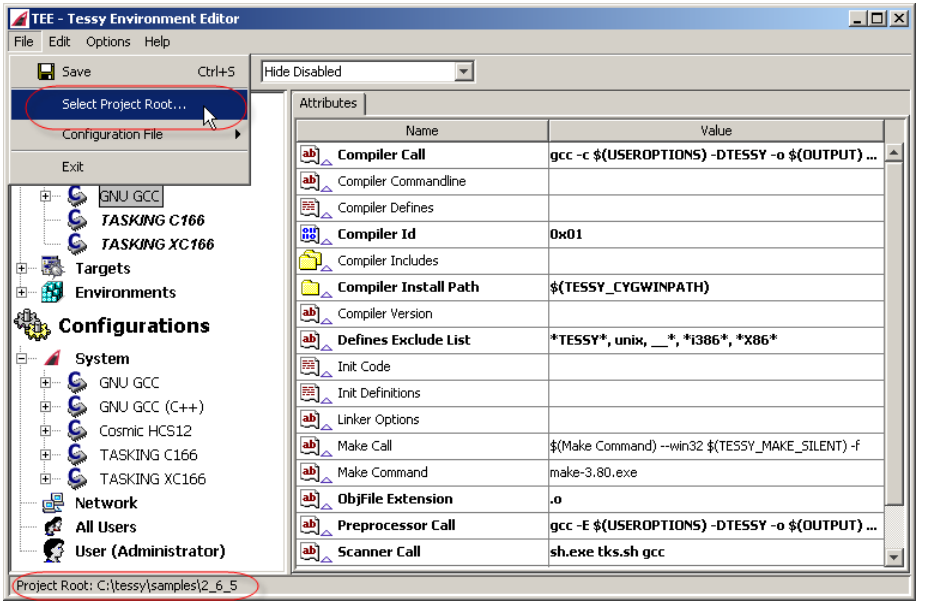
You may create and use a specific **Configuration File** for each project database. This is useful, if you want to share the environment settings with other members of your development team. The configuration file should be stored within your project root directory together with other project related files (e.g. source files).

Such a configuration file may contain only the compiler/target environments you want to use. All other environment configurations are not visible for the user as long as this file is assigned to a given project database (see Assign a Configuration File).

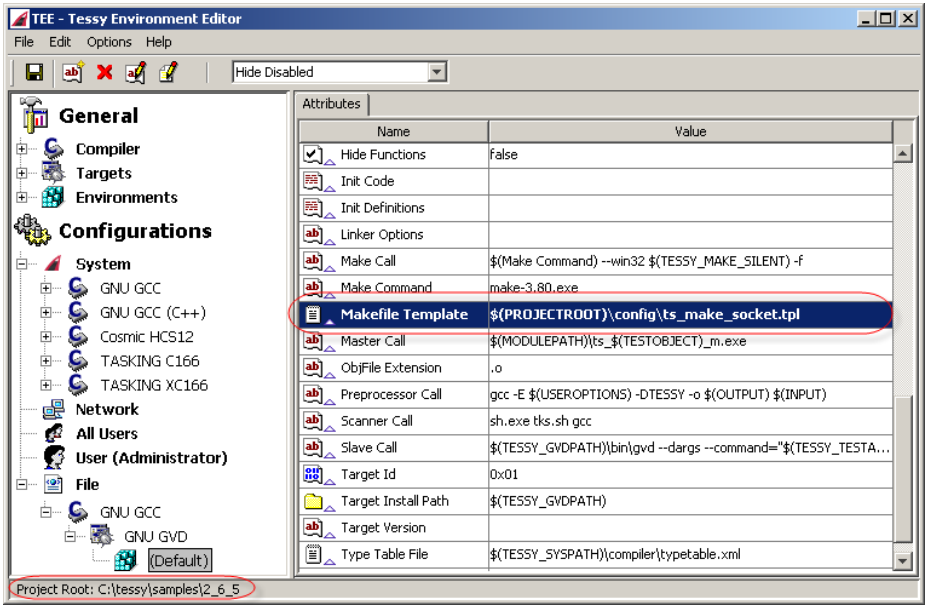
Create a New Custom Configuration File

We recommend using a so called **PROJECTROOT** for your configurations to be able to use relative paths within your configuration file (s. The Project Root). You may specify a temporary **PROJECTROOT** in **TEE** for the configuration of your test environment.

You may start **TEE** from within Tessy by using the **Options|Edit Environment…** menu if a project database is open. This ensures that **TEE** uses the so called **PROJECTROOT** from the project database if specified (s. Assign a Project Root).



We would also recommend storing all specific files (e.g. linker file, makefile template, libraries) for a given custom configuration file in a separate folder other than Tessy's default folders together with other project related files (e.g. source files).



In the example above, the file **`ts_make_socket.tpl`** will be stored in the **`$(PROJECTROOT)\config`** folder.

This ensures that no file will be overwritten if you install a new version of Tessy. You may store these files also in the version control system together with other project related files.

Please mind *that you probably have to revise some files manually when you install a new version of Tessy, e.g. makefile template.*

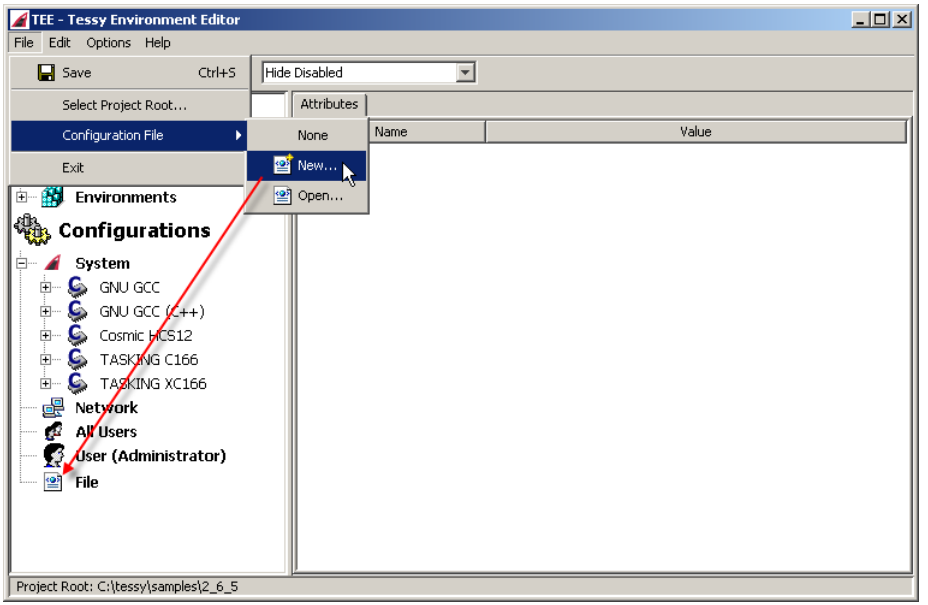
Steps to perform

Create

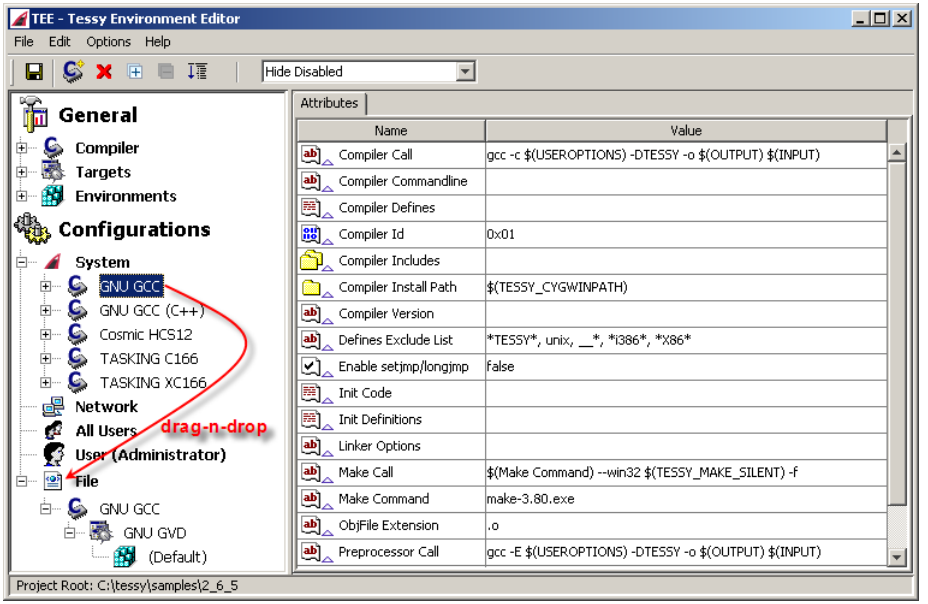
To create a new custom configuration file:

- Choose **File|Configuration File|New...**

  TEE will insert a new **File** node within the **Configurations** section.


TEE - Tessy Environment Editor
File
Edit
Options
Help
Save
Ctrl+S
Select Project Root...
Configuration File
Exit
Hide Disabled
Attributes
Name
Value
None
New...
Open...
Environments
Configurations
System
GNU GCC
GNU GCC (C++)
Cosmic HCS12
TASKING C166
TASKING XC166
Network
All Users
User (Administrator)
File
Project Root: C:\tessy\samples\2_6_5

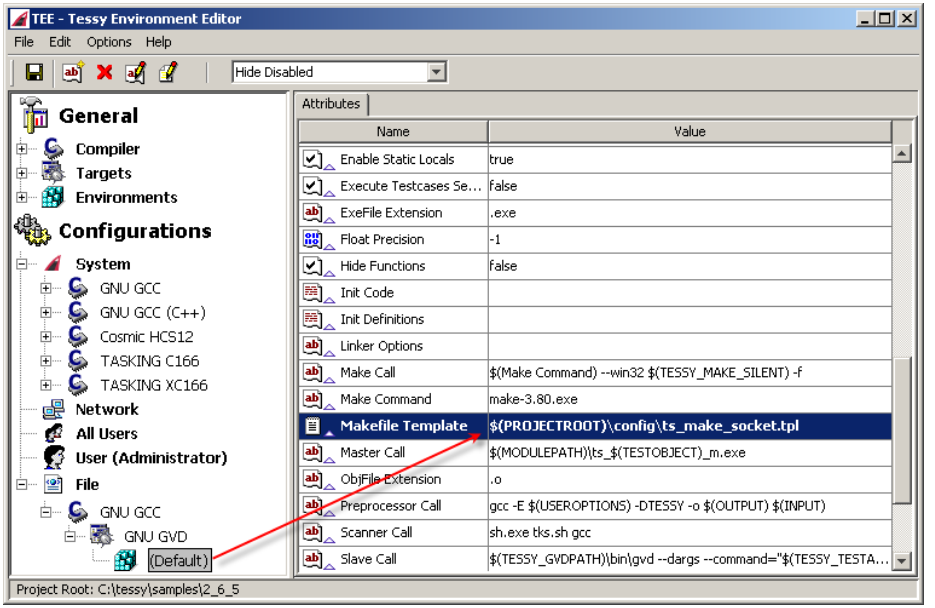
- You may now copy an already created configuration environment from the **Configurations::System** section onto the **File** node by simply using drag-n-drop.



Change

- You may now change settings for this new configuration file to suit your needs.

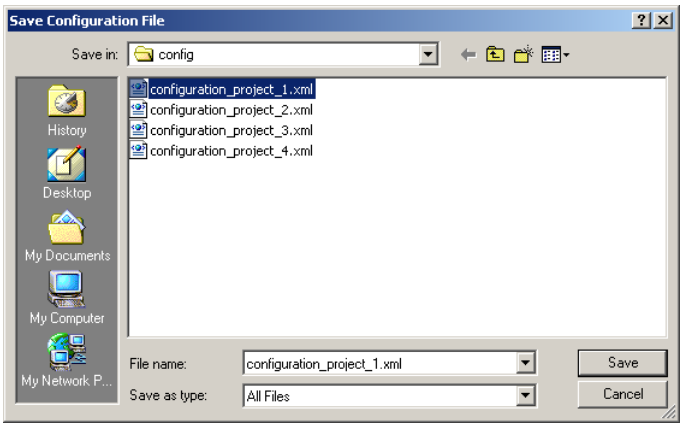
***Please mind**: Carry out all changes for your specific test environment in the **File** section. Only these changes are taken over in the configuration file!*



Save

- Save the changes by using **File|Save**.

  TEE will open the **Save Configuration File** dialog.

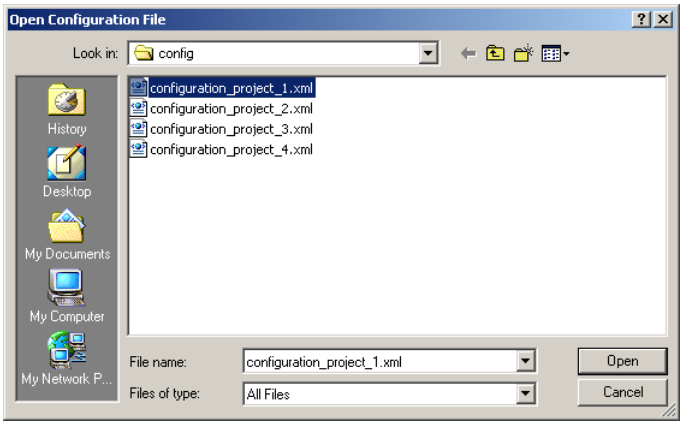


You may assign this configuration file to a project database of Tessy (s. Assign a Configuration File). All the specific environments of this file will then be available with this project database.

Open an Existing File

- If you want to check or change an existing configuration file, please choose **File|Configuration File|Open.**

  TEE will open the **Open Configuration File** dialog.

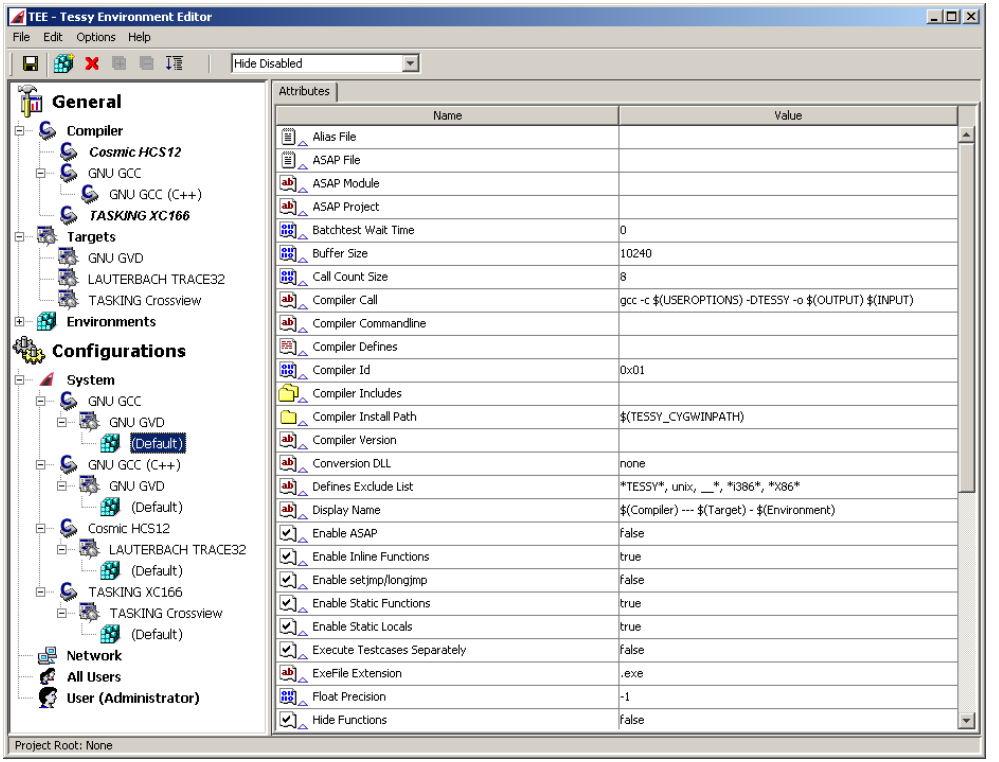


You may start **TEE** from within Tessy by using the **Options|Edit Environment…** menu if you have already assigned a custom configuration file to your project database. TEE will automatically start this file. This also ensures that **TEE** uses the so called **PROJECTROOT** from the project database if specified (s. Assign a Project Root).

- Save your changes by using **File|Save**.

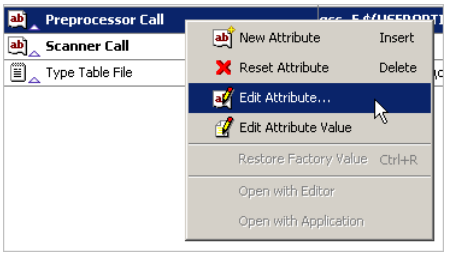
Adding / Editing Attributes

The right-hand **Attributes** pane of TEE displays all attributes of a configuration as soon as an item was selected in the left pane.

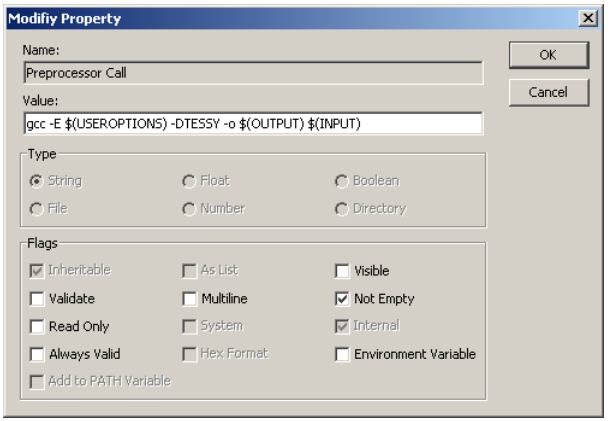


Edit Attribute

- Right-click an attribute and choose **Edit Attribute** from the context menu.



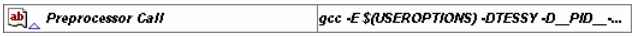
The Modify Property dialog will open.



- If you want to change only an attribute value, choose **Edit Attribute Value** from the context menu.

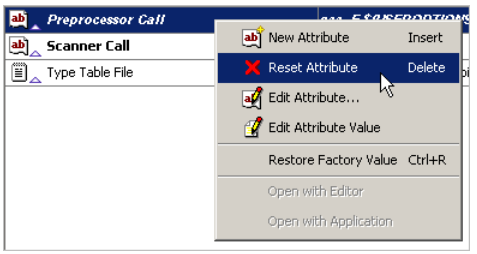
  Depending on the attribute type, either a standard selection dialog for that kind of information will appear (e.g. **Browse for Folder** in case of directories) or the inline editor for the **Value** will be activated.

- If you have changed a default value (other than the factory setting), the attribute will be displayed in italics.



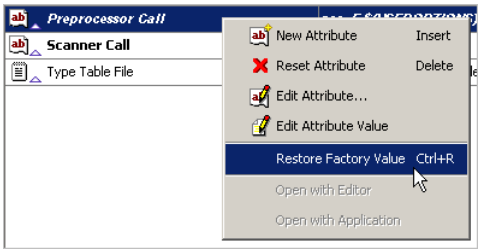
- If you want to delete the attribute value, choose **Reset Attribute** from the context menu or use `[Del]`.

  This will either remove the local value and show the inherited value or delete the whole attribute entry, if it is only defined locally in this section.



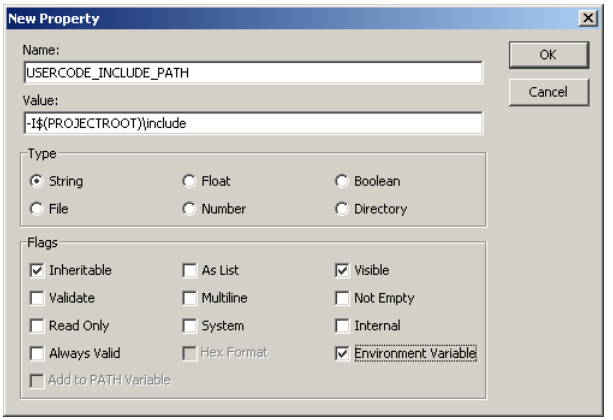
Factory Settings

TEE preserves all default settings. If necessary, you may always use **Restore Factory Value** to revert to the default value.



Add Attribute

- Right-click in the Attributes pane and choose **New Attribute** from the context menu. *The New Property dialog will open.*



Attribute Properties

There are different attribute **Types** available: `String, Boolean, Number, Float, File` and `Directory`. Please make sure to specify the desired type before pressing the **OK** button. The type may not be changed, once the attribute is created.

Additionally, every **Type** may have specific attribute flags. This depends on the **Type** used.

The following table gives a brief description of these flags:

Flags	Descriptions
Inheritable	This flag will always be ticked by default. It controls the inheritance of the attribute: The attribute will be available in all (child) section nodes. *Some basic attributes are defined at the main nodes, e.g.* ***Compiler****. Each supported compiler will inherit these basic attributes.*
Validate	This flag may be important for **Directory** or **File** types. The attribute value will be validated by TEE (e.g. whether the path respectively the file is available). *An error sign will indicate that the file or directory could not be found.*
Read Only	This flag makes it impossible to change a default value by using the attribute pane of the module properties dialog.
Always Valid	*internal flag of TEE*
As List	Using this flag, the attribute value will be handled as list of values (comma separated). The values may be edited using a special list dialog. This is useful for **File** or **Directory** types.
Multiline	Provides a text window for **Multiline** editing.
System	*internal flag of TEE.*
Hex Format	This flag is useful in combination with the **Number** type. TEE will convert all inputs (i.e. a decimal value) to a hex value, e.g. 1 > 0x01.
Visible	This flag makes the attribute visible in the attribute pane of the module properties dialog (and within the test report).
Not Empty	Checks whether the value is not empty. *An error sign will indicate that the attribute does not have a value.*
Internal	*internal flag of TEE*

Environment Variable	This flag is useful during test execution and during the make process: Tessy will create an environment variable within the process space of the process that will be used for test execution (e.g. running the slave process) and for make (e.g. building the test driver). *Note: The environment variable will only be recognized by Tessy if a + sign is used in front of the Make Call value.* *+$(Make Command)*
Add to PATH Variable	This flag is useful for attributes of type **Directory**. Like described above for the Environment Variable flag, the respective directory value will be added to the PATH variable of the process space used for test execution and make.

Symbol Appearance

Following table shows the symbols and their meaning which are used by **TEE**. The attribute icons in front of the attribute names in the right pane, or the icons in the left pane of the **TEE** may have additional symbols attached to accentuate special settings or alert you in case of problems.

Exambles:

	Compiler is disabled.
	The specified directory will be added to the path variable and to the execution environment. The attribute has been inherited.

Table of the symbols which are used by the TEE:

Symbol	Description
	Item is added to the execution environment of Tessy.
	Item is added to the PATH variable of the execution environment.
	Item is disabled
	Item is disabled and another item overwrites (or replaces) this item within the configuration hierarchy.
	Indicates an error
	Indicates an error, but the cause for the error is somewhere within the attributes inherited by this item. You should check for errors upwards or downwards within the configuration hierarchy and fix the error there.
	Information
	Indicates, that the information comes from another item upwards or downwards within the configuration hierarchy.
	Attribute is inherited
	Compiler is active.
	Compiler is inactive (probably overwritten or replaced by another item downwards the configuration hierarchy).
	Environment is active.
	Environment is inactive (probably overwritten or replaced by another item downwards the configuration hierarchy).
	Target is active.
	Target is inactive (probably overwritten or replaced by another item downwards the configuration hierarchy).
	Indicates a warning that there are errors within the attributes of that item (that need to be resolved somewhere else within the configuration hierarchy).
	Indicates a warning, but the cause for the warning is somewhere within the attributes inherited by this item. You should check for warnings and errors upwards within the configuration hierarchy and fix the errors there.

Representation of Values in TEE

The attributes and values in the **Configuration** section are normally inherited from the respective **General** section (Compiler, Targets and Environment). TEE will mark the respective icon with a triangle to indicate that.

TEE will display the values and attributes in different fonts to indicate the following situations:

Normal letters	Represents the factory settings respectively default settings from the **General** sections and have been inherited.
Bold face	The value has been defined the first time for the attribute.
Italic	The factory setting respectively a default setting from the **General** section has been overwritten.

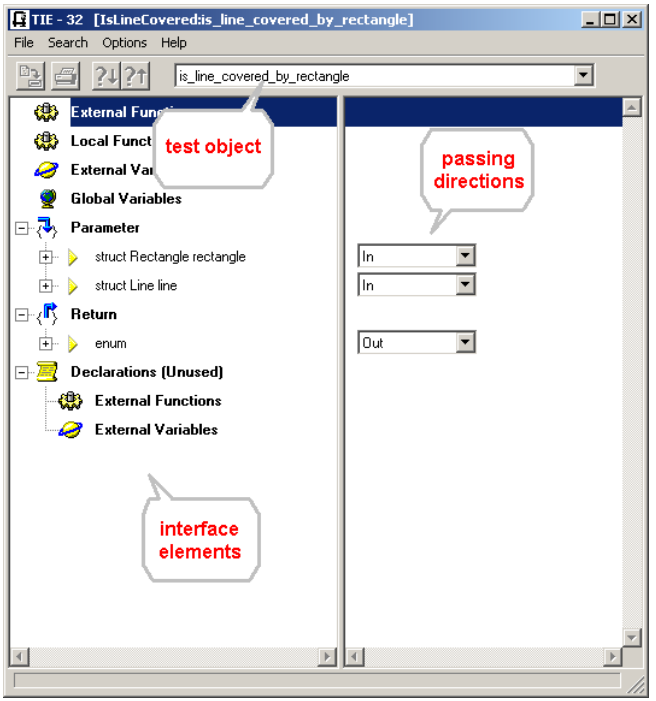
Test Preparation

The Test Interface Editor TIE

After configuring the test environment of a module you have to define the interface information for every test object using the Test Interface Editor **TIE**.

The **TIE** works like a browser. You can navigate through complex interface elements (e.g. structures) down to the basic data types, specifying settings.

The interface information comprise to determine the passing directions of interface elements (e.g. input or output values) as well as arithmetic values, e.g. array dimensions or values of enumeration constants.



Input values are all interface elements that have to be set before execution of a test object. Output values are compared with the expected values after test execution.

Note: *For your convenience, Tessy tries to analyze all interface objects while opening the module. If no "unresolved interface object" has been found, Tessy will open the test object clamp to indicate that (s. Automatic Analysis of the Passing Directions).*

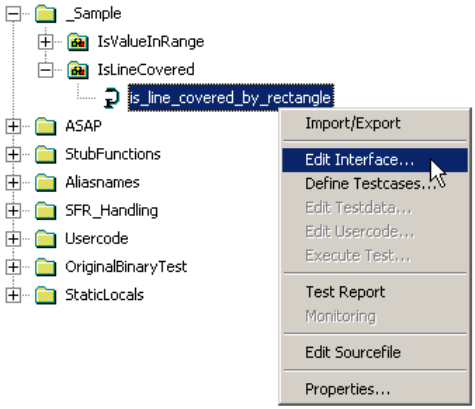
In so far as Tessy has not automatically set all interface information, you have to set them manually or you can change the default settings to your needs (see Open the TIE).

Open the TIE

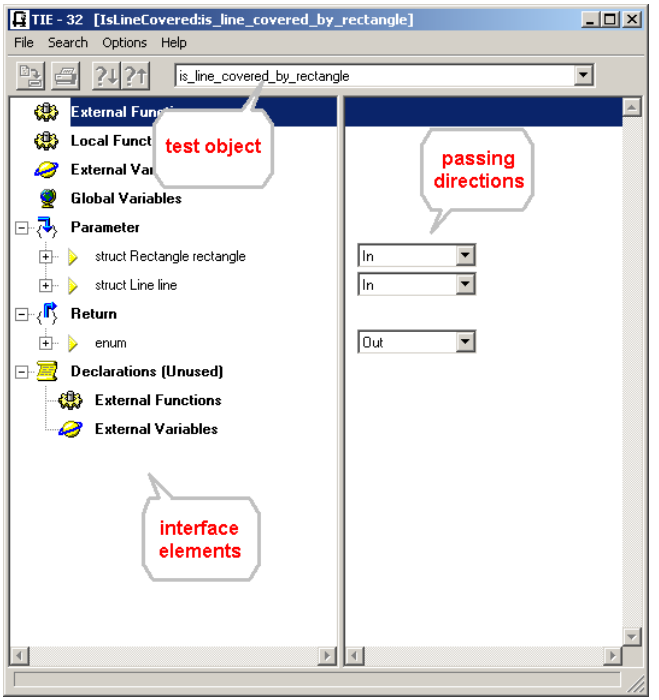
To open the **TIE** do the following:

- Select your test object for which you want to edit the interface information.

- Choose **Edit Interface...** from the context menu or click the respective icon from the tool bar.



The TIE will open.



The **TIE** window is vertically divided into two parts:

- The left pane displays all interface elements of your test object. You can view them and browse down to the basic data types (see Browse the Interface).
- The right pane displays the respective combo-boxes or edit fields to enter the passing directions of variables as well as additional information (see Set Passing Directions).

Browse the Interface

TIE classifies all recognized interface elements of the test object into eight sections as follows:

External Functions	Only declared in the source file and called from the test object.
Local Functions	Functions defined in the source file and called from the test object.

External Variables	Variables only declared in the source or header files and referenced by the test object.
Global Variables	Variables defined in the source file and referenced by the test object. This section also contains global static variables and static local variables defined within functions,
Parameter	Parameters of the test object
Return	Return value of the test object
Enums	Enums which are used by the test object.
Declarations (Unused)	External functions or variables that are declared and used in the source file but not by the test object itself.

Each section node contains their respective interface elements (if any). You can open a section to browse through the interface tree.

***Note**: You may expand/collapse all items of a subtree if you right-click a section or one other node in the tree.*

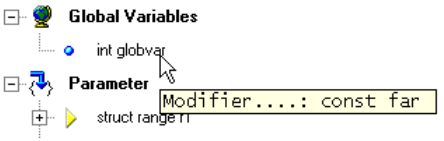
The elements of different levels can be recognized by their differing indentations as well as their structure symbols explained in the table below.

Icon	Description
	This icon indicates that a further level exists. The interface element in question is of a complex C type.
	This icon represents the browsed open state of a complex C type. The sub elements of this type are represented below and indented by one level.
	This icon indicates a basic type that cannot be browsed open.

View Type Modifier

Type modifier, such as far, near, const (etc.) can be displayed within the TIE as follows:

- Press `[Shift]` and move the mouse pointer over a variable. *The type modifier will be displayed as tool tip (if any).*



Special Global Variables

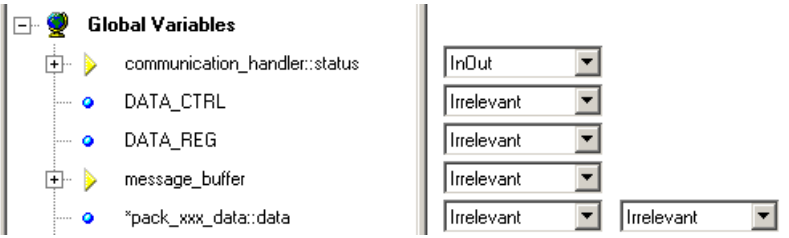
As mentioned above, the **Global Variables** section contains variables with the following lexical scope:

- Defined within the source file
- Defined **static** within the source file
- Locally defined **static** within the **scope of a function** or at the beginning of a statement block

In case of the first two bullets, there will be no difference in the display of the variables. In the last case of static local variables, the name of the variable will be the function name followed by double colons and the variable name (in order to distinguish identically named local variables within different functions).

Example:

```
pack_xxx_data::data
```



If there are nested static local variable definitions within the same function, the names will be enumerated automatically.

Set Passing Directions

The passing direction reflects the kind of usage for each variable while testing the test object.

You can specify how Tessy treats a value for an interface variable either to provide the value before test execution (In) or to keep the value for evaluation and reporting after test execution (Out).

You have to specify the following passing directions for each interface element, if you want to:

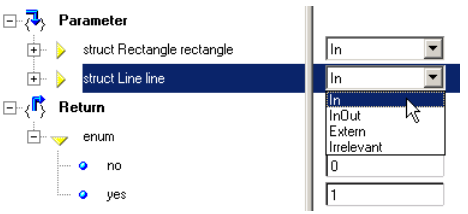
- provide a value for that interface element, because the element is only read by the test object (`In`),
- evaluate and report the results of that interface element, because the element is only written by the test object (`Out`),
- both provide a value and evaluate the result, because the interface element is both read and written by the test object (`InOut`),
- provide a value within the UCE (Usercode Editor) of Tessy (`Extern`) . With this setting, the interface element is visible in the scope of the user code and may be set using C code,
- not use the interface element at all (`Irrelevant`). In this case, you will not see this variable for all further testing activities.

Choose Passing Directions

The passing directions are displayed in the **right** part of the main window.

To set a passing direction of a variable, do the following:

- Choose a suitable passing direction from the combo box list next to the variable, e.g. **In**.



The following table shows possible passing directions of the different types of interface elements.

Interface element	Passing directions				
	In	Out	InOut	Extern	Irrelevant
External Variables	X	X	X	X	X
Global Variables	X	X	X	X	X
Parameter	X			X	X
Return		X		X	X

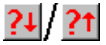
Automatic Analysis of the Passing Directions

During processing of the source files when opening the module, Tessy analyzes the passing directions automatically and stores its findings in the interface database. This information is available in the TIE as default values of the passing directions.

Tessy analyzes the usage of individual interface elements by the test object. Depending on that usage, the following passing directions will be set as default:

- read only :: `In`
- write only :: `Out`
- read and write :: `InOut`
- not used :: `Irrelevant`

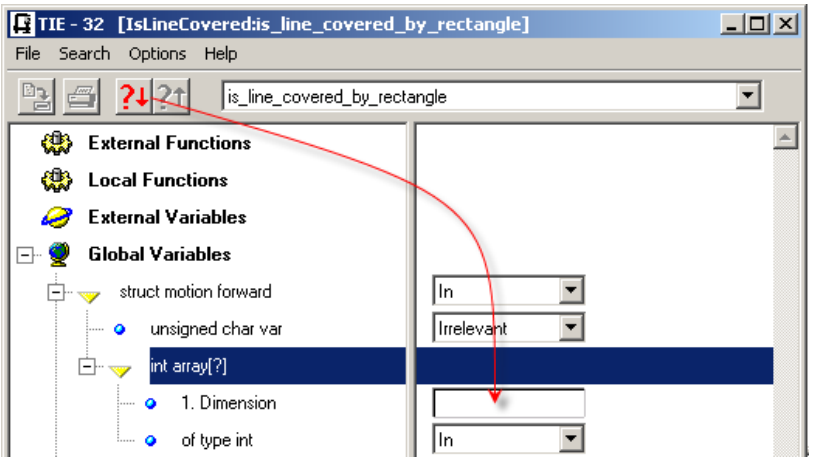
Note: *The clamp (the test object icon) will be opened in case that no "unresolved interface objects" have been found. You may insert test cases directly without opening the TIE.*



Using the buttons **Next unknown object** or **Prev unknown object** from the toolbar will navigate you to either the previous or next interface element that has to be set.

In case that the passing directions or any other interface information could not be determined the respective fields in the TIE will be empty.

For instance: If Tessy could not calculate the size of an array dimension (indicated with a question mark), you have to set them manually.



***Note**: Although Tessy usually correctly recognizes all interface settings, you need to open the **TIE** for every test object to make sure that the value are set correctly or does match your needs.*

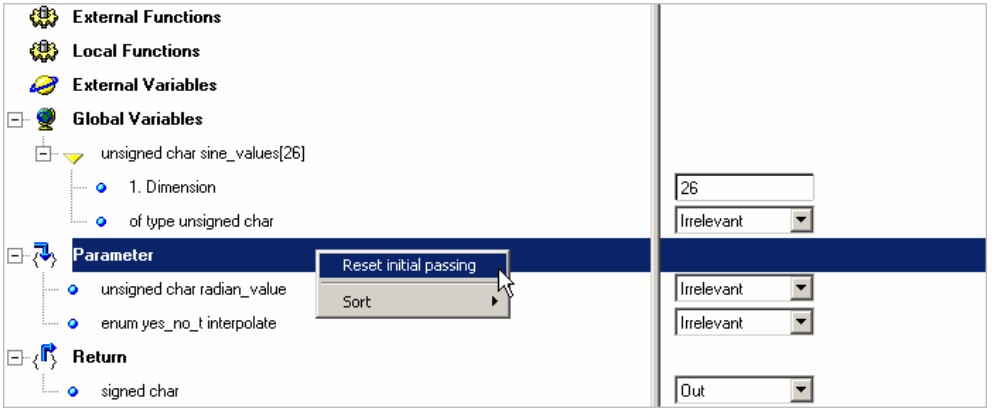
Reset Passing Direction to the Initial Value

Tessy analyzes the usage of individual interface elements by the test object. You may change the passing direction of an interface element to suite your needs (s. Set Passing Directions on page 157).

In case you want to go back to the default value of the passing direction you may reset the passing direction to the initial value by using **Reset initial passing**.

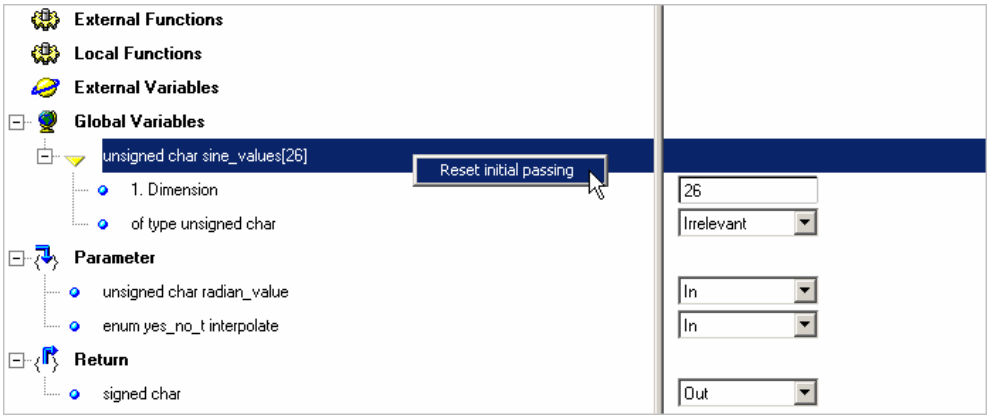
- You may reset the passing direction for all interface elements of one **Section**.

  Select the respective **Section** and choose **Reset initial passing** from the context menu.



- You may also reset the passing direction only for an individual **Interface Element**.

  Select the respective **Interface Element** and choose **Reset initial passing** from the context menu.



Passing Direction of Special Data Types

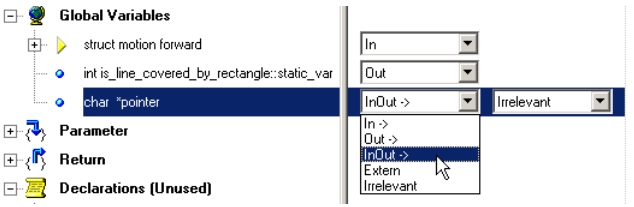
Pointers and complex data types will be treated slightly different as normal data types.

Pointers

The type (Parameter, Global, and Return) and the C data type of an interface element influence the possible passing direction.

In principle one passing direction is defined for every interface element, other than **Pointers**. Both the passing direction of the **pointer** itself and the passing direction of the **target** to which it points have to be specified.

The TIE offers two combo boxes to set the passing directions for pointers.



- Using the left combo box will set the passing direction of the pointer itself and is represented by a pointer symbol, e.g. `In ->`.
- Using the right combo box will set the passing direction for the target.

The passing direction of the pointer and the target may be set interdependently, but they are checked or corrected by **TIE** to ensure valid combinations.

Complex Data Types

Complex data types as **Structure** and **Union** have a dependency between their passing direction of the overall structure/union and the passing directions of their components.

To avoid invalid combinations the **TIE** checks the setting of passing directions for these data types in the following manner:

- When the passing direction of one component is set **TIE** determines the resulting passing direction for the overall structure/union and set them automatically.

- When the passing direction for the overall structure is set, all components are automatically set to the same passing direction.

Arrays

The passing direction of the data type **Array** will be set for the entire to the same direction.

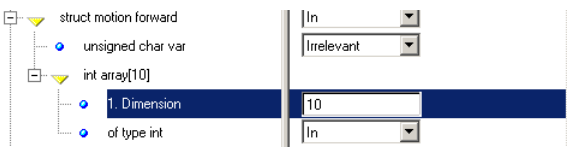
For the data type **Array** only one passing direction will be defined for the whole array. All array elements have the same passing direction.

If the array is made up of structured array elements (e.g. structures) it is possible to define different passing directions for the components of these structures.

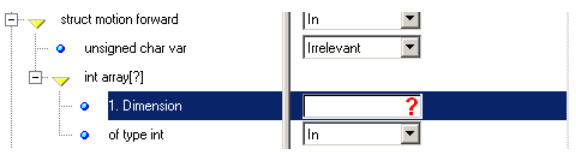
Enter Array Dimensions and Constants

For test execution the information on data types of the test object interface has to be complete. The dimensions for **Arrays**, the values of the enumeration constants for **Enumerations**, and the dimensions for **Bitfields** have to be defined.

If these values have been automatically recognized by Tessy while opening the module, the respective text field will show the calculated value for every data type. In this case, there is no need to changed these values.



If a value for an interface element has not been recognized automatically, the respective text field will be empty or contain the value `-1`. In case of arrays **TIE** will also use question marks to indicate that, e.g. `array[?]`. In all those cases you have to add values manually.



***Note**: Wrong array dimensions or wrong values for enumeration constants can lead the test object to crash during test execution! TIE cannot check for plausibility of used values!*

Define Stubs for Functions

The **TIE** displays all functions used by the test object either in section **External Functions** or **Local Functions**.

TIE provides an interface to define **Stubs** for these functions that will be executed instead of the original function.

TIE distinct two different Stub Functions:

- A stub function for which you may enter C code by using the **Usercode Editor** of Tessy (see Usercode).

  :: Create Stub Function

- A so called advanced stub function that allows to provide values for parameters and return values of stub functions like normal variables in the **TDE**.

  :: Using Advanced Stub Functions

You may define stubs globally for all test objects of the module or you may create a stub independently of the global module setting.

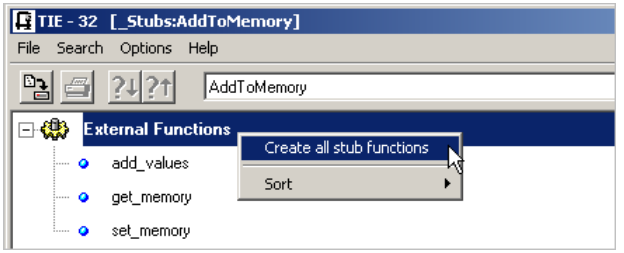
***Note**: For more information about the usage of stub functions please refer to application notes* Using Stub Functions *from the **Help|Documents** menu.*

Create Stub Function

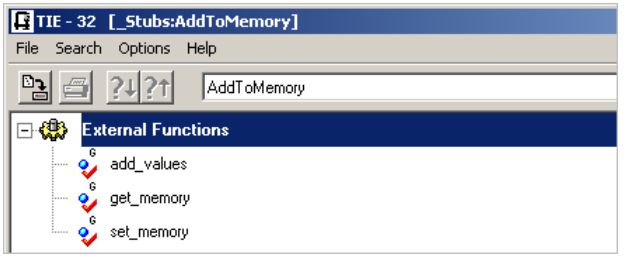
You may create stubs either for **External** or **Local** functions which will be executed instead of the original functions. There are several options available:

- **Create stubs for all functions at once for all test objects of the module (global setting).**

  Select section **External Functions** or **Local Function** and choose **Create all stub functions** from the context menu.

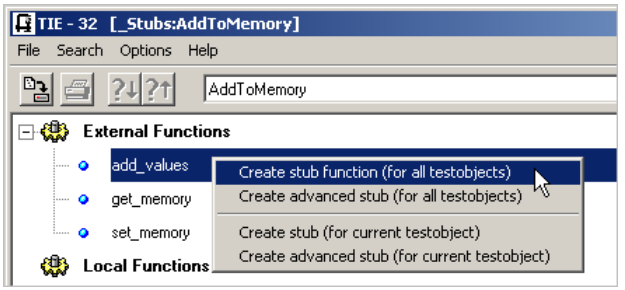


  *TIE will create stubs for all displayed External Functions and indicate that by red ticks. By default, TIE will also define this stubs for all other test objects in the module and indicate that with **G**(lobal).*

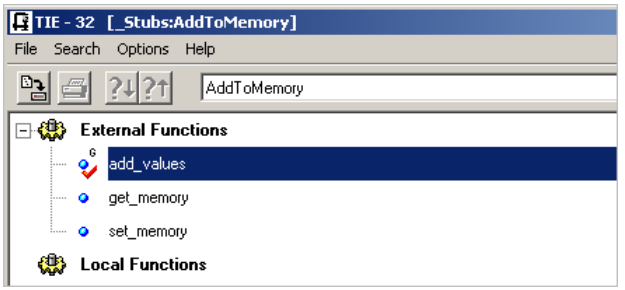


- **Create a stub for a single function for all test objects of the module (global setting).**

  Select one function in section **External Functions** or **Local Function.** From the context menu, choose **Create stub function (for all testobjects).**



*TIE will create a stub for the selected function and indicate that by a red tick. TIE will also define this stub for all other test objects in the module and indicate that with **G**(lobal).*

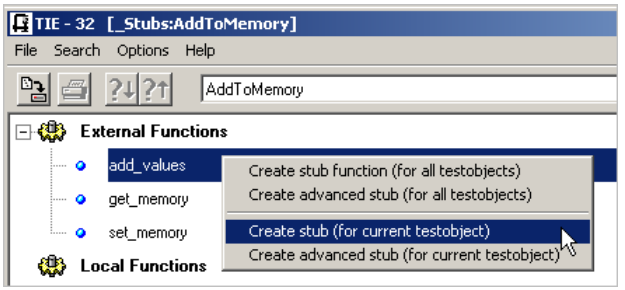


- **Create stub for the current test object**

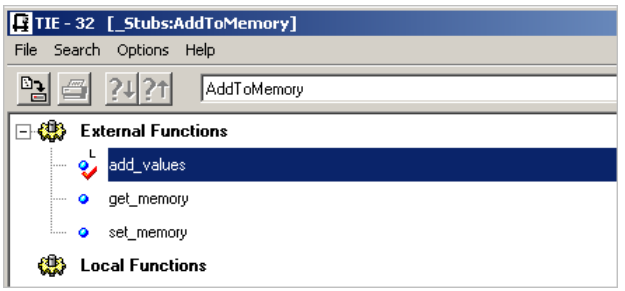
  Example :: `AddToMemory`

  This option enables you to create a stub independently of the global setting of the module.

  Select one function in section **External Functions** or **Local Function.** From the context menu, choose **Create stub function (for current testobject).**



TIE will create a stub for function `add_value` *for the current test object* `AddToMemory`*. This stub will work independently of the global module setting. A red tick marked with* ***L****(ocal) will indicate that a stub has been created only for the current test object.*

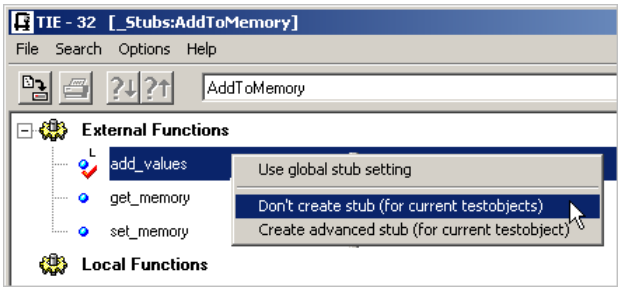


- **Change current settings for the current test object**

  Example :: `AddToMemory`

  There are two options available: You may return to the global setting of the function (**Use global stub setting**) or you want to create no stub for the current test object (**Don't create Stub (for current testobject)**). The last case leaves the global setting of the function untouched.

  In the latter case, TIE will remove the stub for the current function and will indicate that by clearing the red tick.



Please mind, *that you have to provide C code for the defined stub functions by using the* ***Usercode Editor*** *of Tessy (see Usercode), e.g. a return value.*

Using Advanced Stub Functions

This enhancement to normal stub functions (s. Create Stub Function) allows to provide values for parameters and return values of stub functions like normal variables. (see The Test Data Editor **TDE**)

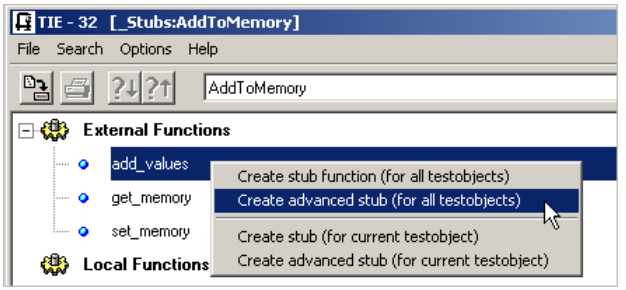
You may check if the stub is called by the test object with the specified parameter values, otherwise the test fails. You may also provide a return value for further processing by the test object. This reveals if the test object handles the return values of the stub function correctly.

Create Advanced Stub Function

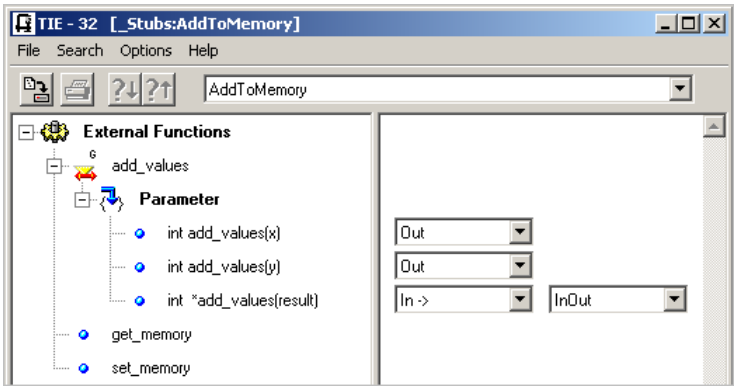
You may create advanced stubs for **External** and **Local** functions. There are several options available:

- Create an advanced stub for a function for all test objects of the module (global setting).

  Select a function either from section **External Function** or **Local Function** and choose **Create advanced stub function** from the context menu.



*The function will then be marked by a red double arrow and a **G** to indicate the global setting of the advanced stub.*



You may set the passing directions for parameters and the return value.

***Note**: Remark the slight difference in passing direction semantics as follows:*

- The passing direction **Out** for a parameter means, that the parameter of the stub function will be checked in the event of calling the stub function from your test object (e.g. to check, if the test object calls the stub with a specified value).

  The passing direction **In** for the return value means, that you need to provide the value to be returned by the stub function when called from your test object.

  A pointer need to have the passing direction **In**, if you want to return a value within the pointer target (which would result in passing direction **In** for the target).

  If you want to check the pointer value of a parameter (e.g. if the pointer targets a

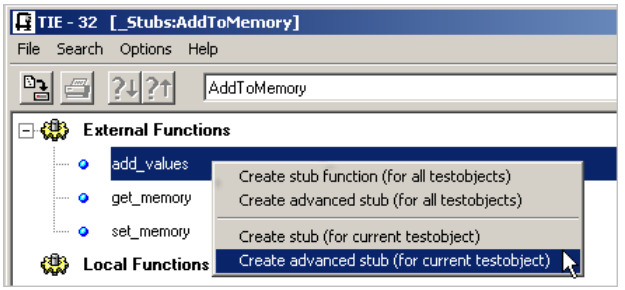
specific variable within the test object's interface), then you need to set the passing direction to **Out**. In this case, no further handling of the pointer target will be possible (passing direction **Irrelevant** for the target).

- **Create an advanced stub for the current test object**

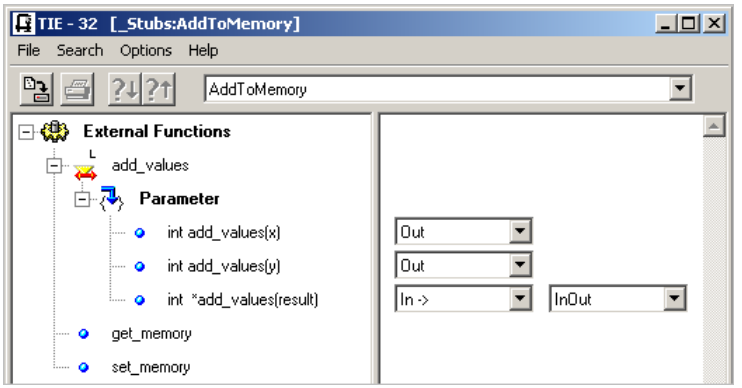
  Example :: `AddToMemory`

  This option enables you to create an advanced stub only for the current test object. This case leaves the global setting of this function for all other test objects of the module untouched.

  Select one function in section **External Functions** or **Local Function.** From the context menu, choose **Create advanced stub function (for current testobject).**



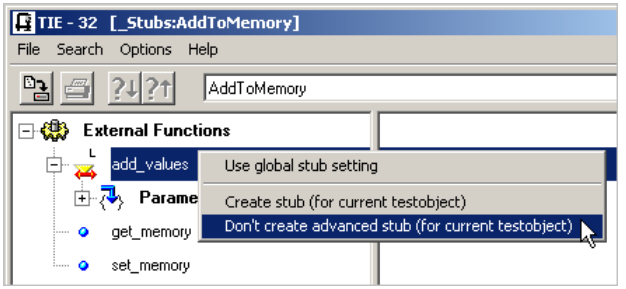
*TIE will create an advanced stub for function `add_value` for the current test object `AddToMemory`. This advanced stub will work independently of the global module setting. A red double arrow with **L**(ocal) will indicate that an advanced stub has been created only for the current test object.*



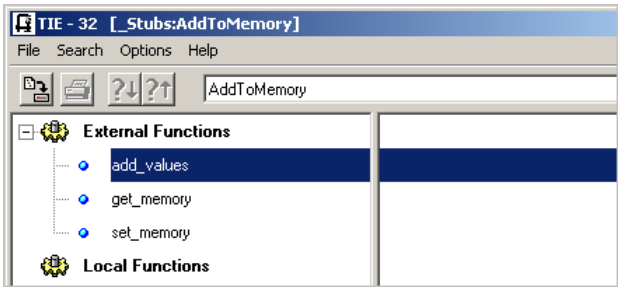
- **Change current settings for the current test object**

  Examble :: `AddToMemory`

  There are two options available: You may return to the global setting of the function (**Use global stub setting**) or you want to create no stub for the current test object (**Don't create Stub (for current testobject)**). The last case leaves the global setting of the function untouched.



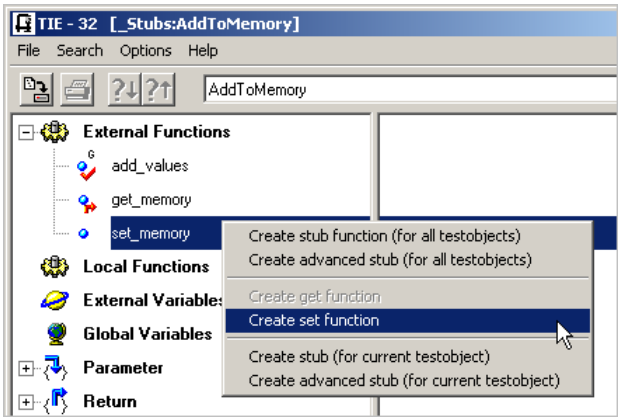
TIE will remove the stub for the current function and will indicate that by clearing the red tick. This will work independently of the global module setting.



Using Get/Set Functions

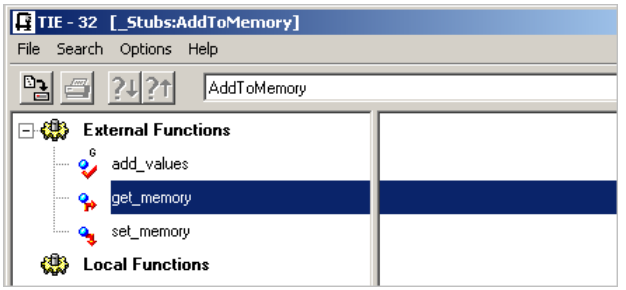
Get/set functions have additional entries in the context menu, allowing to either define a stub function or to use a get/set function referencing a generated variable instead.

- **To use a get/set function, you need to select one of the get/set entries in the context menu.**



The defined **Get Function** will then be marked by a red arrow pointing up-right.

The defined **Set Function** will be marked by a red arrow pointing right-down.



If you don't want a get/set function to be defined, do the following:

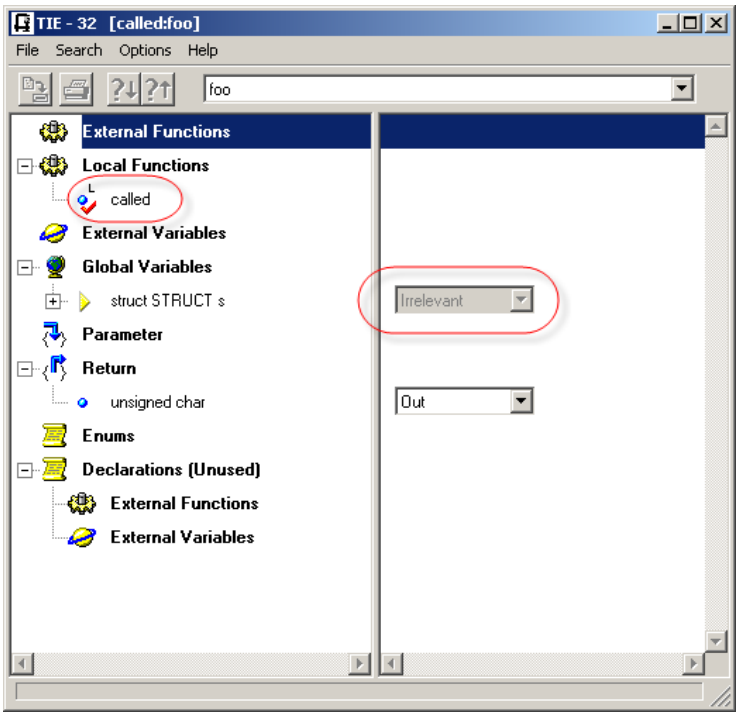
- To remove a get/set function

  select the function and choose **Don't create get/set function** from the context menu.

  TIE will remove the get/set function and indicate that by clearing the red arrow.

Hide Global Variables for Local Stub Functions

This feature will be automatically available as soon as you define a normal stub or an advanced stub for a local function (s. Define Stubs for Functions on page 163). TIE will set the passing direction on `[Irrelevant]` for the global variables which are used by the stubbed function. The passing direction cannot be changed any more. The combo box has been disabled.



Define or Declare External Variables

The global variables used by the test object are distinguished by the **TIE** according to whether they are defined or only declared in the source file, e.g.

- Defined: `float result;`
- only declared: `extern int motion_flag;`

Defined variables are displayed in section **Global Variables** while declared variables appear in section **External Variables**.

Besides specifying the passing directions for external variables, you can also define these variables using TIE.

- **To define all external variables at once**

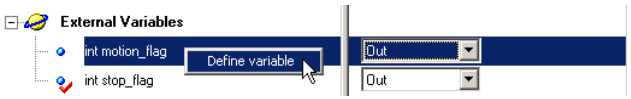
  select section **External Variables** and choose **Define all variables** from the context menu.

TIE will define all variables and indicate that by red ticks.

- **To define a single external variable**

  select one external variable from section **External Variables** and choose **Define variables** from the context menu.

  TIE will define the variable and indicate that by a red tick.



If you don't want define variables, do the following:

- **To remove all defined variables**

  select section **External Variables** and choose **Don't define all variables** from the context menu.

  TIE will remove all defined variables and indicate that by clearing the red ticks.

- **To remove a single defined variable**

  select one external variable and choose **Don't define variable** from the context menu.

  TIE will remove the defined variable and indicate that by clearing the red tick.

Important note: *Tessy will only define the variables according to their type. A default initialization of the values of the variables does not take place during definition but is dependent on the used compiler.*

Define Missing Linker References

If your source code (respectively module) contains more than one test object, it might be possible that the other test objects use external references, e.g. variables and function calls, which however are not used by the current test object.

All variables and functions referenced in the source file but not used by the current test object will be displayed in section **Declarations (Unused)**.

If these references cannot be resolved by libraries or object files added to the linker options, you have to define them by **TIE** in order to build up the test driver successfully.

- **To define unused declarations, do the following:**

  Open section **Declaration (Unused)** and select either section **External Variables** or **External Functions.**

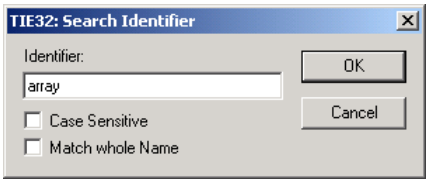
  The steps you have to perform are nearly the same as described in Create Stub Function on page 164 and Define or Declare External Variables on page 173. You may define unused functions or variables or not define them.

Search for Variables

If the interface of your test object is more complicated you may use the **Search** dialog of **TIE**.

- Either choose **Search Identifier** from the **Edit** menu or use `[Ctrl]+[F]`.

  The Search Identifier dialog will open.



- Enter your search term and choose following options if necessary:

  - Case Sensitive
  - Match whole name

- Click **OK**.

Save Interface Information and Exit the TIE

- To save your settings either choose **Save** from the **File** menu or click the respective icon from the tool bar.
- To close the TIE either choose **Exit** from the **File** menu or use `[Alt]+[F4]`.

Edit Test Cases

Enter Test Cases

The next step of the test preparation will be the definition of test cases. One test case includes at least one test step to hold the test data for test execution.

- Please refer to section The Test Data Editor TDE.

If suitable, you can also specify user code to be executed at specified places during the test execution, e.g. c code for defined stub functions

- Please refer to section Usercode.

The systematical approach to create test cases is to use the **Classification Tree Method**. The distribution of Tessy includes the specialized Classification Tree Editor **CTE** to perform the specification of test cases.

The CTE has a direct interface to Tessy and allows to assign values for variables of the test object interface within the classification tree. You can export all test items and the assigned test data into the test database of Tessy.

- To find out how to access the CTE from Tessy refer to the Section "Systematic Test Definition".

You may also create test cases manually by using Tessy.

- Please refer to Section "Insert Test Cases Manually".

Systematic Test Definition

The systematic test definition takes place with the help of the Classification Tree Method by using the Classification Tree Editor **CTE**.

The interface between Tessy and CTE has been extended to provide a list of input and output variables to CTE that may be assigned to elements (classifications and classes) of the classification tree. This allows to assign values for variables of the test object interface within the tree. If such a tree element is selected for a test case within the combination table, the respective value will be used for export to Tessy.

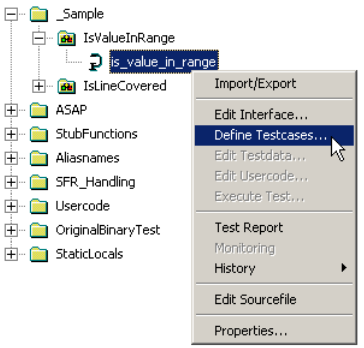
***Important note**: You must start CTE by Tessy if you want to use this feature (s. Open the CTE).*

Open the CTE

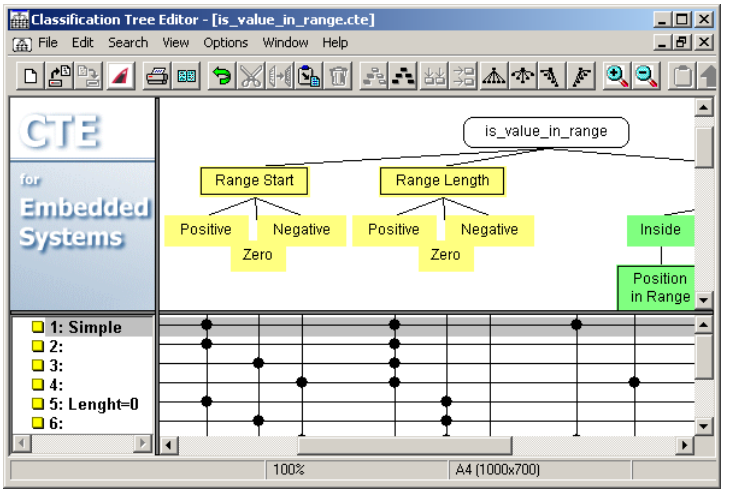
***Note**: From the naming convention used within CTE we define the more common term **test item** to name test cases, test steps and test sequences.*

To open the CTE, do the following:

- Select the test object and choose **Define Testcases** from the context menu or click the respective icon from the toolbar.



The CTE will open.



***Note**: Please refer to the CTE User Manual for more details on using the CTE.*

***Important note**: If the test object interface has still unresolved interface objects, Tessy will not open the test object clamp to indicate that. Neither the interface information will be available for CTE nor can transfer you test cases to Tessy. In that case, please open the test interface editor TIE to update the status.*

Assign Variables

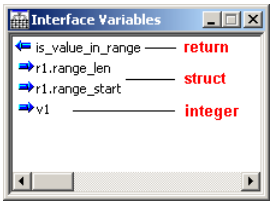
Tessy provides a list of input and output variables to CTE that may be assigned to classifications and classes of the classification tree.

Important note: *It's necessary to start CTE by Tessy to use this feature (s. Open the CTE).*

Currently, following interface objects are not supported:
- arrays
- static local variables
- pointer

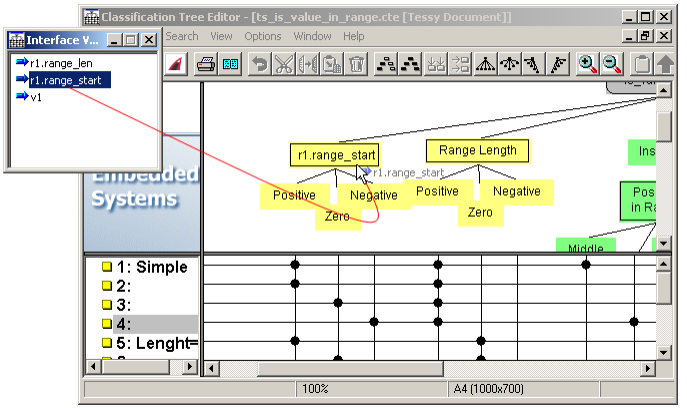
To assign variables:

- Open the *Interface Variables* window by using **View|Variables…**

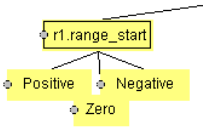


- Drag-n-drop a variable from the *Interface Variables* window onto the classification element, e.g. `Range Start`.

Note: *If you want to use the name of the interface variable as name for your tree element, please press [Shift] and drag-n-drop then the variable onto the classification element. The tree element gets the name of the variable.*



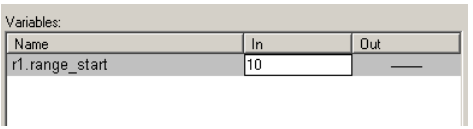
- CTE will mark the respective tree element and all connected child elements by a gray dot. This indicates that a variable has been assigned.



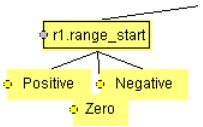
Assign Values to Variables

To assign values:

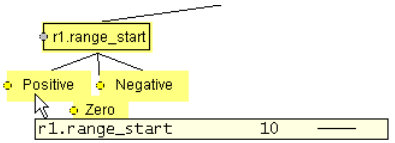
- Right-click a class of the classification tree and choose **Properties…** from the context menu.
- From the *Element Properties* dialog, choose the **Variables** tab.
- You may enter a default value within the **In** or **Out** field. This is dependent from the passing direction of the interface variable.



CTE will change the gray dot to yellow, as soon as a value has been assigned to the class element.

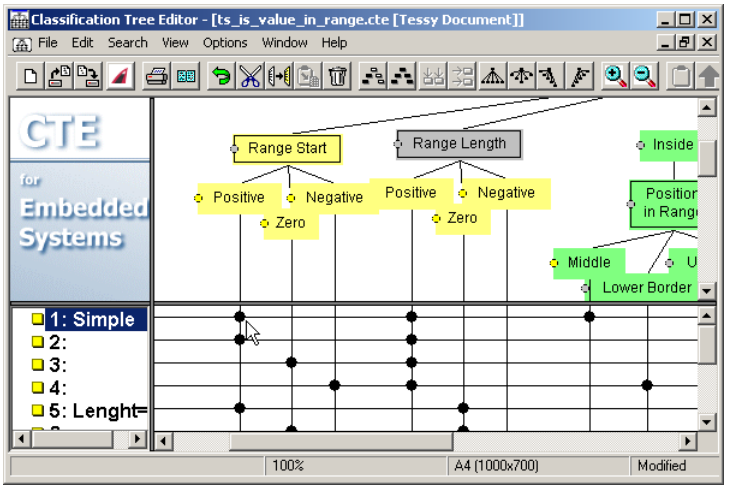


You may move the mouse pointer over the class element to view the assigned value. CTE will display the value using a tool tip.



Note: *If you assign a variable to a classification element, all connected classes will inherit the variable and an assigned default value too. You may assign a specific value for every connected classes. This will then overwrite the default value that has been assigned to a classification (if any). If necessary, you may assign a variable and a value only to a class.*

If a tree element is selected to specify a test case within the combination table, e.g. `Positive` (s. picture below), the respective value will be used for export to Tessy (s. Export Test Items to Tessy).



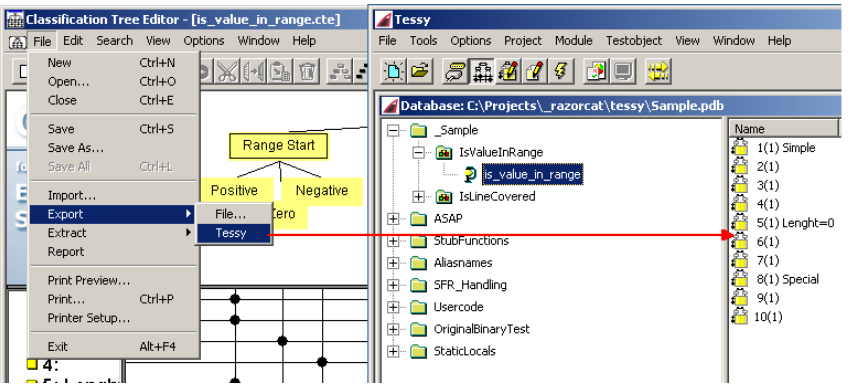
Export Test Items to Tessy

You can directly export all created test items into Tessy's test database. CTE will export all specifications and values for variables if used.

Important Note: *Please make sure that you have the appropriate test object selected within Tessy. CTE will export the test items to the currently selected test object.*

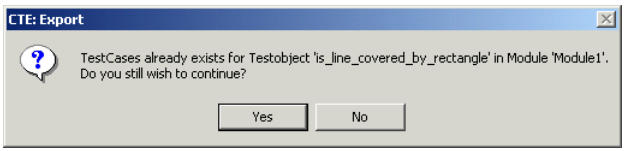
To export test items to Tessy, do the following:

- Choose **Export|Tessy** from the **File** menu or click the respective icon from the toolbar.



Important note*: Exported values for variables will be read-only in Tessy. You cannot change such values by using the TDE (s. chapter The Test Data Editor TDE.). You must use CTE and must then synchronize the test database of Tessy!*

Before Tessy imports all test items into the test database, Tessy will inform you whether test items already exist for this test object. In this case, the following message will appear.



This message indicates that possible inconsistencies may occur when importing the new test items. If, however, only further test cases have been added to the already existing ones, you just have to enter new test data for the new test cases (see The Test Data Editor TDE).

- If you click **Yes**, all test cases will be moved into Tessy's test database. Existing test data will be preserved.
- If you click **No,** you will cancel the operation.

Note: *Each exported CTE **test case** will create a Tessy test case with one test step by default. You may add further test steps within Tessy if required. Each exported CTE **test sequence** will create a Tessy test case with a corresponding number of test steps as of the CTE test sequence. If you want to add further Tessy test steps, you need to add them within CTE and export again.*

Now you can close the CTE. Select **Exit** from the **File** menu. You will now be returned to the main window of Tessy.

Important note*: After exporting of test cases, Tessy will copy the CTE document into the respective module folder and rename then the document in ts_the_testobject_name.cte. The CTE document must be stored in the module folder.*

Synchronize CTE and Tessy

Exported values for variables will be read-only in Tessy. Therefore you cannot change such values by using the TDE (s. chapter The Test Data Editor TDE.). You have to use CTE respectively the underlying CTE document to change exported values.

You have to start CTE from within Tessy, otherwise the interface description of the test object will not be available (s. Open the CTE). CTE loads the appropriate CTE document for the respective test object automatically. The CTE document will always be stored in the module directory.

After you have carried out changes in the CTE document, you must synchronize the test database of Tessy. This mean, you must export the test items a second time (s. Export Test Items to Tessy).

Status of assigned Variables

The status of an assigned variable is indicated by a colored dot on a classification tree element.

Following states are displayed:

Status	Description
gray	A variable was assigned, no value was specified or values are not complete (s. Assign Values to Variables).
yellow	A variable was assigned, a value was specified.
red	CTE was not launched by using Tessy (s. Open the CTE). Or, a variable was assigned but is no longer valid (e.g. has been removed from the source code).

Representation of Test Results

As soon as a test was carried out in Tessy, the results for every test case appear also in the CTE.

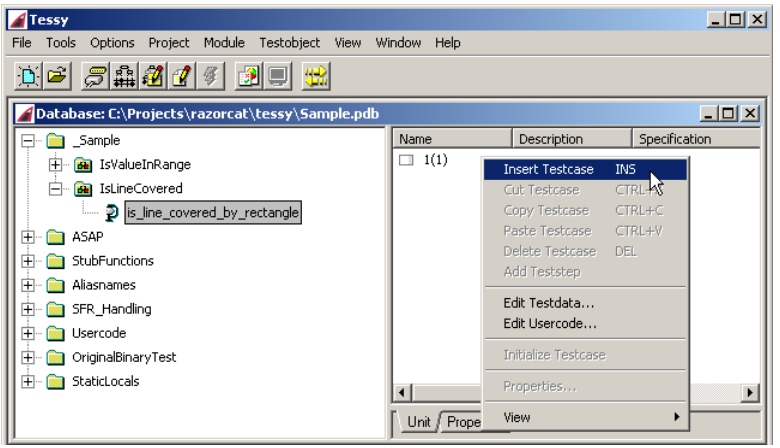
A test case is marked be a green tick if the test has passed. A red cross will indicate that the test has failed.

Insert Test Cases Manually

Tessy provides an interface to create test cases respectively test steps manually without the need to use the Classification Tree Editor **CTE**. This is useful for simple test objects with a few test cases that can be documented in a few words manually.

To create test cases, do the following:

- Select a test object from the module. Choose the unit tab from the right pane.
- Click within the pane and choose **Insert Testcase** from the context menu or press `[Ins]`.



A new test case with one test step will be created.

***Important note**: If the test object interface has still unresolved interface objects, Tessy will not open the test object clamp to indicate that. You will not be able to insert test cases. The "Insert Testcase" context menu option will be grayed out! In that case, please open the test interface editor TIE to update the status.*

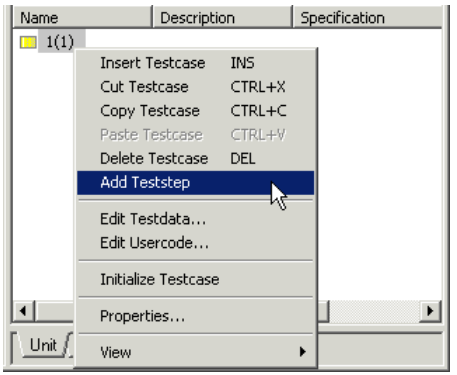
Each test case contain at least one test step. The test step contains the input and output values of a test object. You may add additional test steps to add more test data for the same test context.

Add Test Steps

You can add additional test steps to each test case. Every test step contains a complete set of test data. For instance, the mechanism of test steps can be used to achieve an initialization of the test object before executing the test step that checks the actual test condition of the current test case.

To add a test step to a test case, do the following:

- Select one test case.
- Choose **Add Teststep** from the context menu.
  Tessy will insert a new test step.

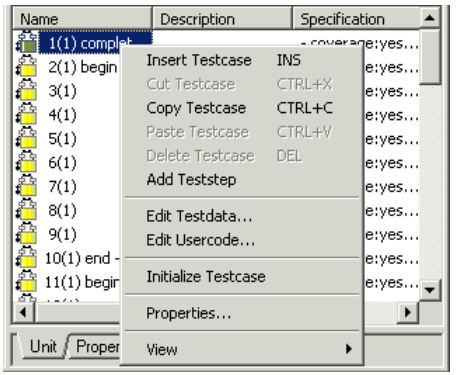


***Note**: You can also create or delete test steps using the **TDE** (s. The Test Data Editor TDE).*

Delete Test Cases

There is a slight difference between test cases imported from CTE and test cases created by Tessy.

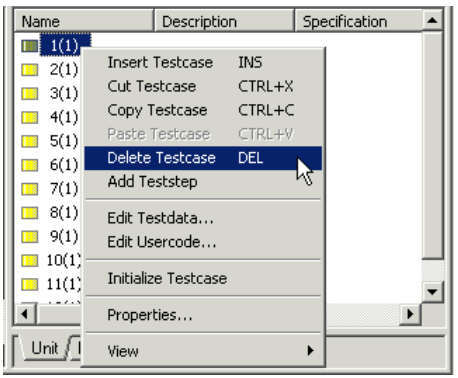
You cannot delete test cases created by CTE using Tessy. This feature has been disabled by default.



For this purpose you must use CTE and reimport them for the respective test object to Tessy. This will prevent possible inconsistencies within the CTE document when deleting test cases by Tessy (see Systematic Test Definition).

This way, you can only delete test cases created by Tessy as follows (see Insert Test Cases Manually):

- Select one or more test cases.
- Choose **Delete Testcase** from the context menu or press `[Del]`.



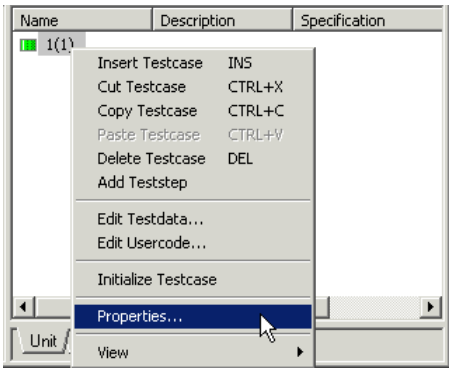
Delete Test Steps

You can delete test steps of a test case using the Test Data Editor **TDE**. Please see "The Test Data Editor TDE" for more information. At least one test step will remain.

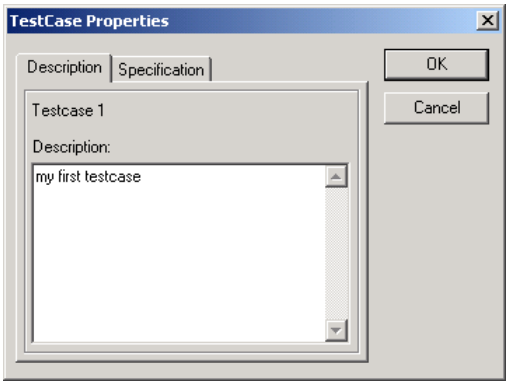
Enter Test Case Description and Specification

You have the option to enter a short description and specification for every manually created test case (see Insert Test Cases Manually.

- Select a test case and choose **Properties...** from the context menu.



The *TestCase Properties* dialog will open.



- Enter a *Description* and a *Specification* for the test case.
- Click **OK**.

Usercode

Tessy provides an interface to specify the so called “Usercode” that will be executed at a certain point in time during the test execution. Using the **Usercode Editor** UCE, you may specify such C code fragments or emulator scripts (depending on the selected target configuration).

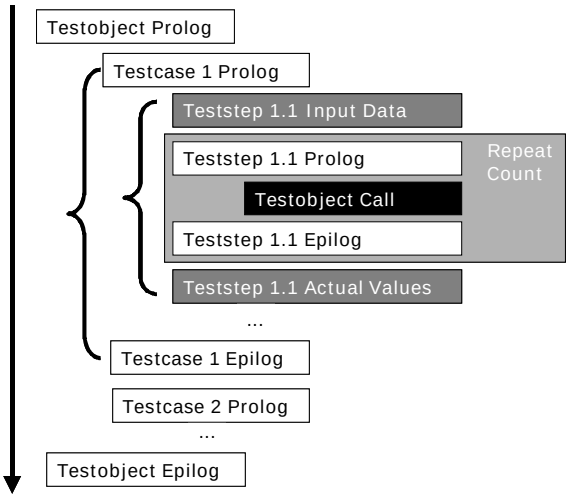
In the C part of the Usercode you may

- **define** and initialize test object specific **variables** like complex structures or arrays, e.g. characteristic curves
- execute code to **initialize variables** of your test object's interface
- define or declare global **helper variables** to be used within the usercode
- enter “`#define`” statements to be used within the usercode
- specify the **number of execution times** for a test step
- **evaluate any kind of variable or memory location** to be of interest for the test report
- trace the actual **number of calls** to all functions called from the test object
- **implement** the code body of **stub functions** called from the test object

The C part of the usercode will be integrated into the test driver and executed at the places specified (e.g. before or after execution of a test case or test step). The Usercode can be executed as follows:

- *Prolog:* execution takes place before the respective element, and
- *Epilog:* is executed after execution of the respective element.

The test object prolog will be executed at the beginning of the test, whereas the test object epilog will be executed at the end of the test. The following figure outlines the call sequence of the user code parts.



The prolog and epilog sections and the repeat count for the test step are edited within UCE. The picture also shows the interaction of the usercode sections with the test data provided within TDE and the actual values that are saved to the test database and evaluated against the expected results within TDE.

During the test object call, the code specified for the stub functions (if any functions are called from your test object) may be executed depending on the code logic of your test object. This is important in order to understand the usage of the eval macros described later in this chapter.

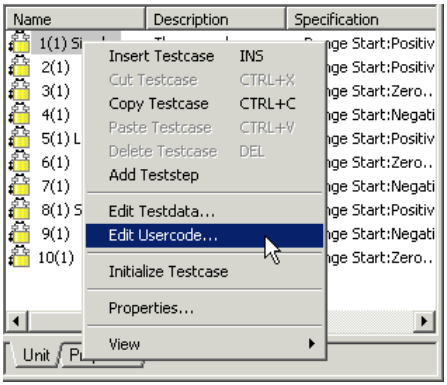
Within the script part of the usercode you may execute script commands in the scripting language of the respective emulator. Currently, only HiTOP and TRACE32 are supported for this feature. Please refer to application notes **Using Usercode** from the **Help|Documents** menu.

Starting the Usercode Editor UCE

***Note**: Before you can open the UCE within Tessy, you have to create at least one test case.*

Please do the following to open the UCE:

- Select one test case and choose **Edit Usercode** from the context menu.

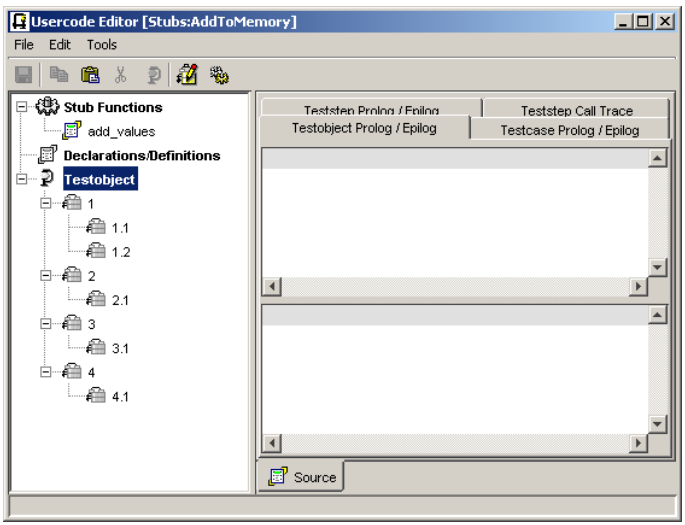


The Usercode editor will open.

You may also start the **UCE** from within the **TDE** (see The Test Data Editor TDE) or choose **Edit Usercode** from the tool bar.

Using the UCE

The UCE consists of a tree on the left hand side showing the Usercode structure containing the available entries and a number of tab panes on the right hand side to edit the Usercode.



Initially, there are only the following structure elements available in the UCE tree:

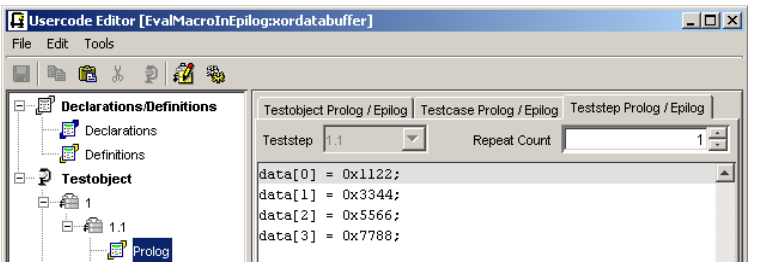
- **Declarations/Definitions** holds the declarations and definitions of helper variables, that you may want to use within prolog/epilog code or stub function implementation.. If selected, two tabs **Declarations** and **Definitions** will appear on the right-side window for editing.
- **Testobject** holds the prolog and epilog for the test object and all test case nodes as children.
- Nodes for each test case and test step

If the test object calls at least one function that should be stubbed, there will be a **Stub Functions** section containing one node for each stub function. Depending on your settings within TIE, this list contains all called functions that you have specified to be defined as stub functions.

When you select a tree node on the left side, the corresponding editor pane will be shown on the right hand side. As soon as you enter any text, a new node will automatically appear at the corresponding place in the tree. If you later want to change the Usercode, simply select the respective entry and the appropriate code fragment will be displayed in the right hand window

Enter C Code

- To edit the prolog or epilog of a test object, select the tab **Testobject Prolog / Epilog** and enter the code into the window.
- If you want to edit the prolog/epilog for a test case/test step select a test case or test step from the tree and enter the code into the right hand window. A new node will automatically appear at the corresponding place in the tree.



- For stub functions, select a function from the tree under **Stub Functions**.

- For declarations or definitions, select **Declarations/Definitions** from the tree. You may add helper variables or definitions of external variables from the test object interface.

Note: *Please enter the* ***definition*** *of global variables under* ***Definitions*** *and not under* ***Declarations*** *because the code under* ***Declarations*** *will later be added to a generated header file and will be included in multiple source files of the generated test driver!*

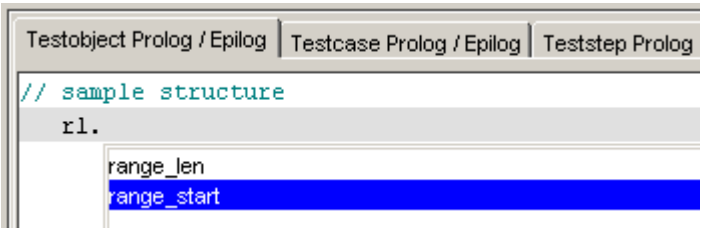
If you have already entered Usercode on a tab, you will automatically arrive at the respective tab by selecting a tree node on the left side.

You can also reach the previous or following prolog or epilog of a test case/test step by using the shortcut `[Ctrl]+[Page up]/[Ctrl]+[Page down]`on your keyboard.

Active Usercode

The UCE provides a popup menu containing all available interface variables for convenient editing. To show this popup, simply press the `[Ctrl]+[Space]` key. You may then choose a variable name from the menu. Pressing the **Esc** key removes the popup.

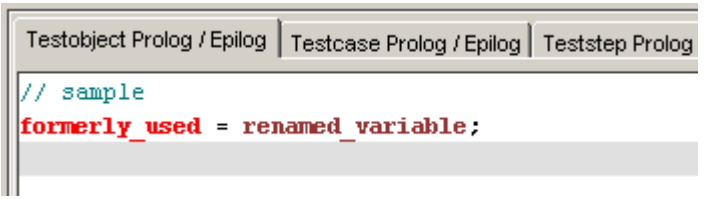
In case of struct or union variables, the respective components will also be displayed, if you have already written the name of the variable directly followed by a dot like shown in the example below.



The UCE popup contains all global variables used by the test object and the predefined variables described later. The variables will be highlighted by UCE to visualize that they are part of the test object interface.

When saving the Usercode, Tessy will replace each occurrence of test object interface variables with the internal ID of this variable. This feature allows to assign changed variable names during reuse operations (see The Interface Data Assign Editor), resulting in automatically changed Usercode in case of changes to the test object interface.

One consequence is that variables used within the Usercode, that are not anymore part of the interface, will be highlighted in ligh red. They need to be changed manually or be removed.



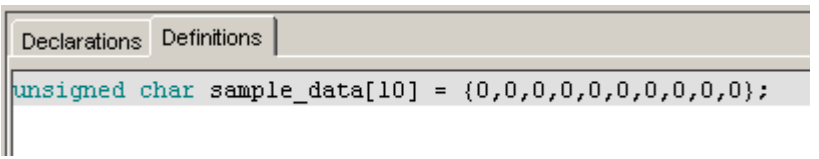
In the above example, the variable *formerly_used* has been part of the interface before the reuse operation (displayed in light red). You would need to replace the variable with any useful statement or delete it completely, if it's obsolete.

The other variable *renamed_variable* has been renamed and is now displayed with the new name (highlighted in dark red to indicate an interface variable).

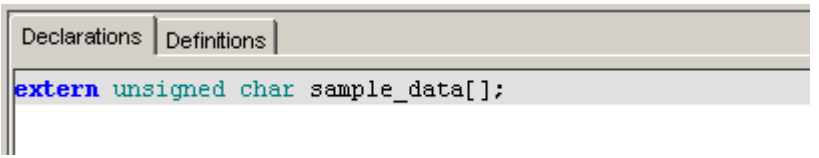
Definitions/Declarations

The definitions section of the Usercode may be used to define helper variables to be referenced within the prolog or epilog or within a stub function. You may also define external variables of the test object interface (sometimes it's easier to use the definition and initializers to setup a variable correctly).

If the variable is only used within the prolog or epilog, you do not need to add a declaration of this variable to the declarations section. That's only required, if the variable shall also be used within stub functions.



The above definition of the variable *sample_data* would require the following declaration because the variable shall be used within a stub function:



The declarations will be included into the generated Usercode files and stub function files. In case of local functions that shall be stubbed, the declarations will also be added at the beginning of the source file (that contains the local function) before compilation.

Important note: *The declarations may conflict with type definitions of the source file, if you are stubbing* ***local*** *functions (as opposed to external functions). If you need to declare conflicting types, enclose the conflicting declarations like described below.*

In order to make declarations only visible within the generated Usercode files, use the following define statement:

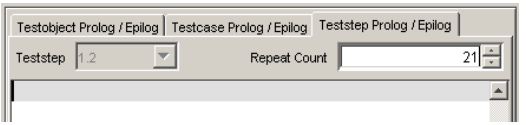
```
#ifdef TESSY_USERCODE
    ... some declarations ...
#endif
```

This define is only active when compiling Usercode parts (e.g. prolog, epilog, stub code for external functions). It's **not** active, when compiling the source files of the module, if the source file contains local functions to be stubbed. This causes any declarations enclosed within the above define statement to be invisible within the respective source file.

Repeat Test Step Prolog/Epilog Execution

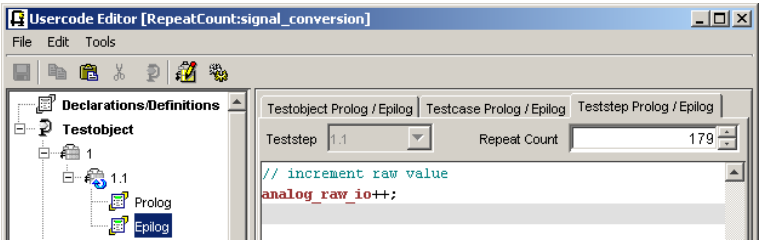
The prolog and epilog may be executed several times using the repeat count field located at the upper right border of the test step tab. Setting the counter to a value greater than one will repeat the following sequence the specified number of times:

- execute the prolog (C code and script code)
- call the test object
- execute the epilog (C code and script code)



This repeat count may be used to execute the test object several times with continuously changing input data like in the example below: The input value signal initially set using the TDE will be incremented for each loop.

Example:



In the same way, you may assign calculated output values of your test object to other input variables within the prolog.

Repeat Test Object Execution

Furthermore, you are provided with a repeat count for the execution of the test object itself (Without executing the prolog/epilog). This would be useful, if you need to execute the test object many times with the same input values to reach a certain state of the test object (e.g. for testing debouncing functions).

Syntax:

```
TS_REPEAT_COUNT= <Number of executions>;
```

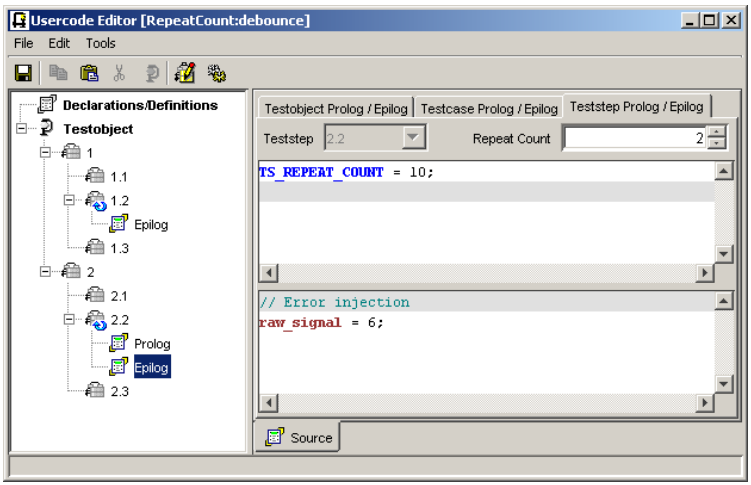
Example:

```
TS_REPEAT_COUNT=3;
```

This means that the test object will be executed three times. The input values to the test object will not be changed for any repetition. The output values will be saved to the test database after the last repeat cycle.

Important note: *The TS_REPEAT_COUNT variable is* ***only*** *available within the* ***test step prolog****.*

Example:



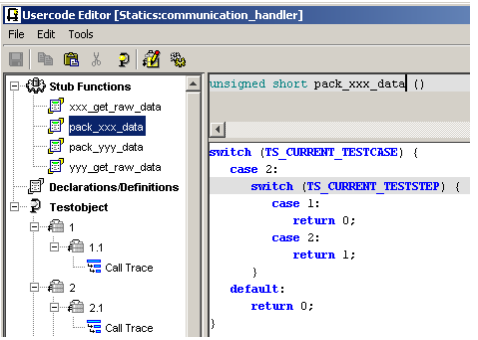
In the above example, the test object will be called 10 times, then the epilog will be executed, which changes the value of one of the input variables. Then, due to the test step repeat count set to 2, the prolog will be executed again and though the test object will be called 10 times again with the new input value.

Current Test Case/Test Step Counter

The predefined variables `TS_CURRENT_TESTCASE` and `TS_CURRENT_TESTSTEP` are available in the scope of the Usercode indicating which test case or test step is currently being executed.

You can use them to execute specific code for each test case or test step. This may be helpful when coding stub functions.

Example:



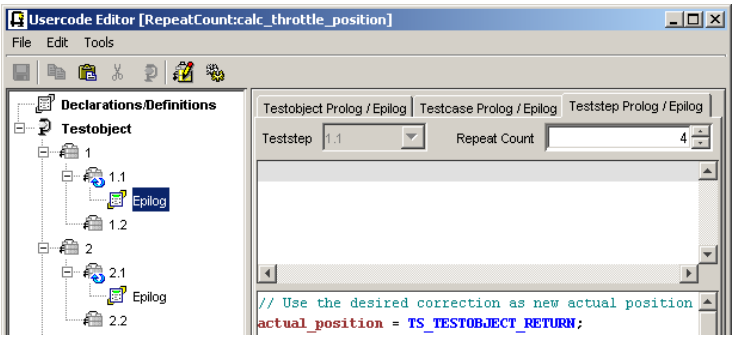
The above implementation of the stub function returns different values depending on the currently executed test case and test step. For more information about the usage of Usercode please refer to the application notes **Using Usercode** from the **Help|Documents** menu.

Assigning the Test Object Return Value

In case you want to calculate something based on the return value of your test object, there is a predefined variable TS_TESTOBJECT_RETURN available.

This variable holds the return value of the last test object call (only if the test object has a return value at all). You may use this variable within the test step epilog as shown below.

Example:



In this example, the return value of the test object is assigned to the input variable *actual_position*. With the test step repeat count set to 15, the test object is called in a sort of closed loop (15 times) with the return value as feedback value.

Important note: *The TS_TESTOBJECT_RETURN variable holds valid data* ***only*** *within the* ***test step epilog****.*

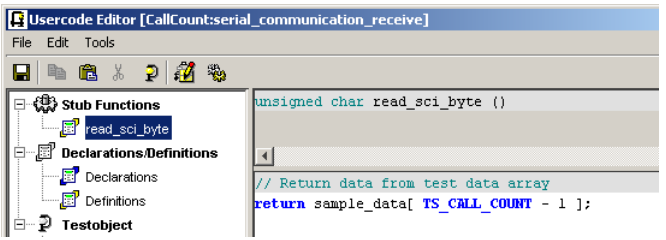
Call Counter for Stub Functions

For the implementation of stub function code, it is sometimes useful to know about the current number of calls to this same function.

The predefined variable TS_CALL_COUNT is available in the scope of each stub function code and counts how many times the function has been called (There is a separate counter for each stub function, if there is more than one function).

The count starts from one, so the first time the stub function is called, TS_CALL_COUNT equals 1.

Example:



In the example above, the stub function returns values from an array using the TS_CALL_COUNT variable as index. Subsequent calls to the stub function will return one element of the array after the other.

Note: *When using the TS_CALL_COUNT as index into an array, don't forget to substract 1, because TS_CALL_COUNT starts with 1 and not with 0 as required for C indices.*

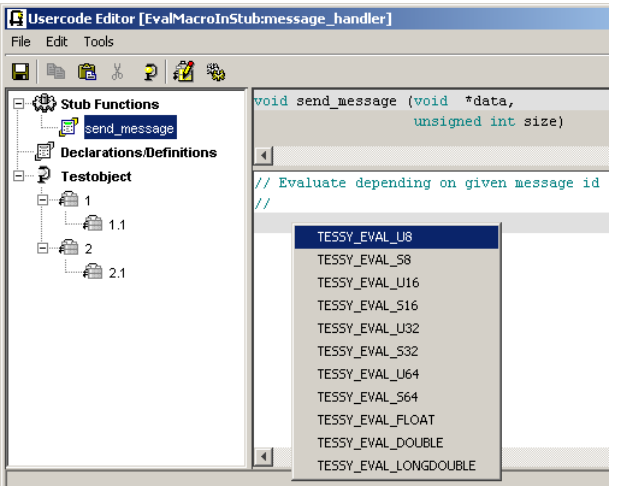
Using Eval Macros

Within the test step epilog or within stub functions, you may evaluate any variable or expression using the Tessy eval macros. These predefined macros allow to check an

expression against an expected value (this may also be an expression). The result is stored within the test report like the evaluation of normal output variables of the test object.

Eval macros may be used within the following Usercode sections:

- test step epilog
- stub function code



A context menu within the above sections shows all available eval macros. When selecting one of the menu entries, the corresponding eval macro with already filled sample arguments will be added at the cursor position.

The only difference of the eval macros is the type of argument for the actual and expected value. See below for a description of the available types.

Syntax:

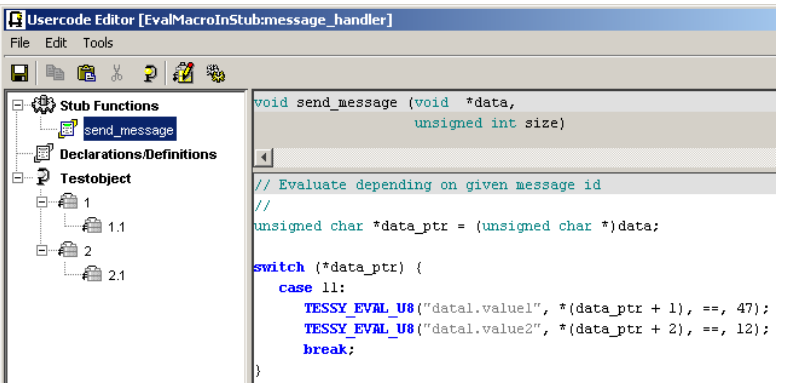
```
TESSY_EVAL_U8("<name>", <actual>, <operator>, <expected>);
```

The actual and expected argument may be any C expression of the respective type and the operator has the standard C semantics. Both may be chosen from the following list:

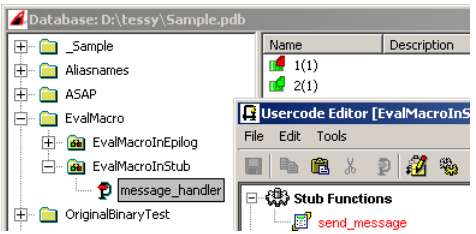
Typename	C Type
U8	Unsigned, 1 byte
S8	Signed, 1 byte
U16	Unsigned, 2 bytes
S16	Signed, 2 bytes
U32	Unsigned, 4 bytes
S32	Signed, 4 bytes
U64	Unsigned, 8 bytes
S64	Signed, 8 bytes
FLOAT	Float
DOUBLE	Double
LONGDOUBLE	Longdouble

Operator	Meaning
==	Equal
!=	Unequal
<	Less
>	Greater
<=	Less or equal
>=	Greater or equal

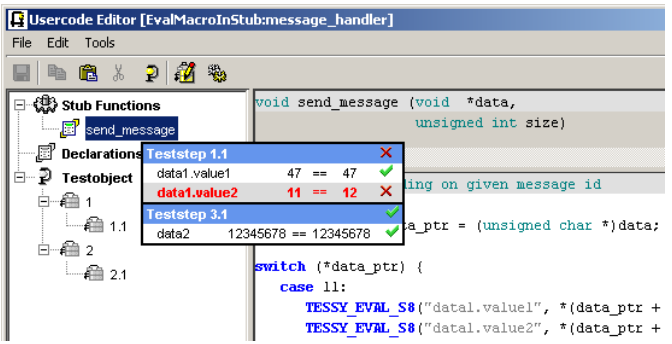
Example:



Each invocation of an eval macro results in an additional entry within the test report. All eval macros will be added to the list of actual/expected values of the current test step. You may also view the results of eval macros within UCE after executing the test like shown below:

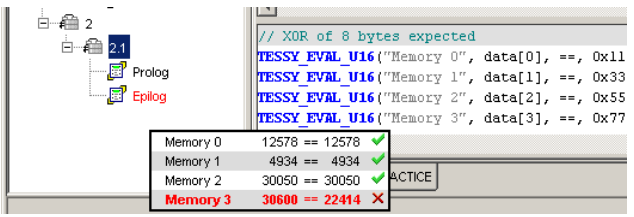


The red marked stub function node visualizes a negative evaluation result of one of the eval macros executed within this stub function. If you select the stub function node or move the mouse over the node, a tooltip containing the eval macro results will be shown.



The eval macro results in the tooltip are grouped by test step for better overview. If the stub function has not been called or the eval macro has not been executed while running a certain test step, there will be no entry for this test step within the tooltip.

The same result tooltip is also available for the test step epilog. A red marked epilog node indicates errors in the eval macros like shown below:



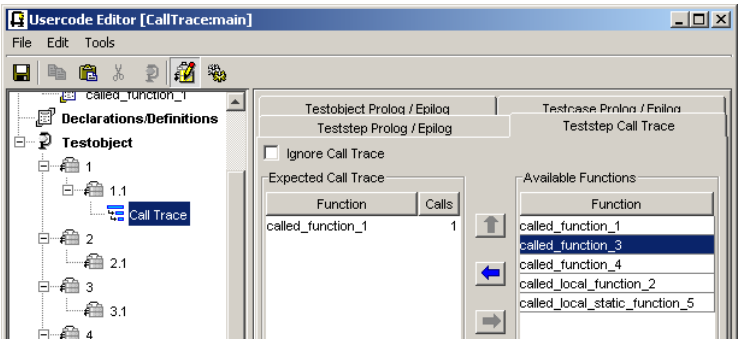
Note: *You have to use braces {} to delimit Eval Macros if you use if/else statements.*

Example:

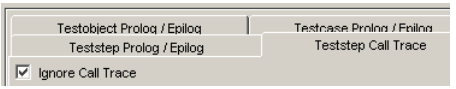
```
if (x) {
  Tessy_Eval_Macro (…);
}
else {
  Tessy_Eval_Macro (…);
}
```

Call Trace

The call trace feature of Tessy allows to check the calling sequence of functions called from the test object. For each test step, you may check if a specified sequence of functions has been called. If the test object calls at least one function, the **Teststep Call Trace** tab of UCE will be visible.

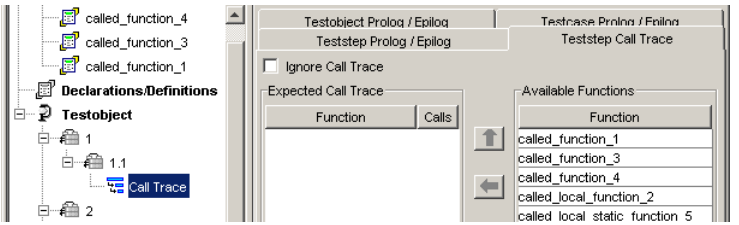


You may edit the desired sequence of calls by dragging function entries from the right list of **Available Functions** into the left list **Expected Call Trace**. You may also use the arrow buttons to move list elements from right to left, up and down and vice versa. Subsequent calls of the same function will be combined automatically to result in one function list entry with the **Calls** number being incremented.



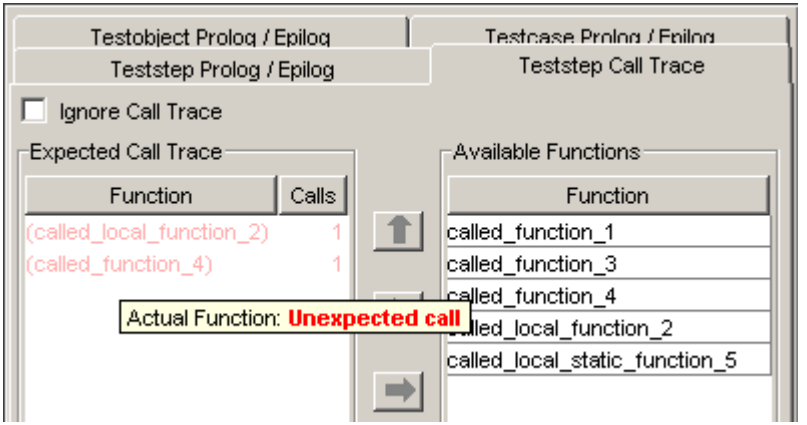
By default, the tick **Ignore Call Trace** is set. This indicates, that the call trace shall be ignored completely. As soon as you add any function to the expected list, this tick will be removed. If you want to ignore the call trace again, you may select the tick and all functions will be deleted from the expected list.

Whenever the **Ignore Call Trace** tick is removed, there will be a new **Call Trace** node within the UCE tree on the left hand side. You may verify, if no function at all is called from your test object by removing the tick and leaving the **Expected Call Trace** list empty as shown below.



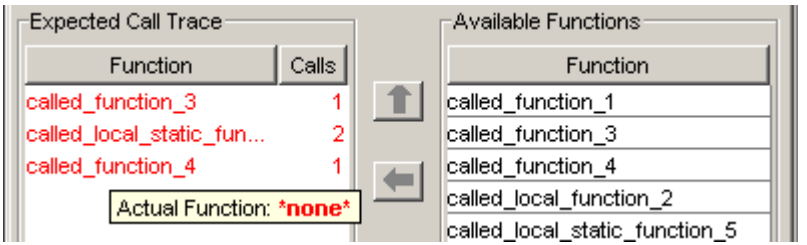
The results of the call trace will be stored within the report and also visualized within UCE after execution of the test object. A red marked **Call Trace** node indicates an error. By selecting the node, you will see the **Teststep Call Trace** tab showing one of the following error possibilities:

Example: Empty call trace expected, but functions were called



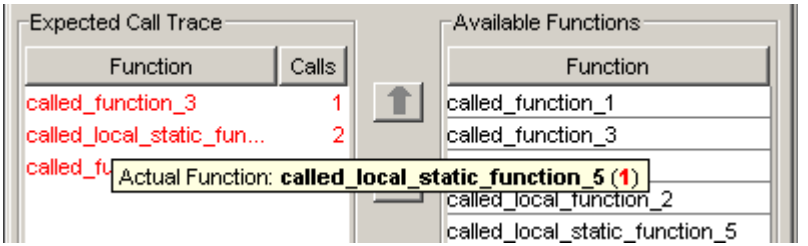
In this case, the tooltip indicates that the called functions are *Unexpected calls* with respect to the empty expected call trace. Such function calls are visualized in pink and enclosed in parentheses.

Example: Expected function call, but no function was called



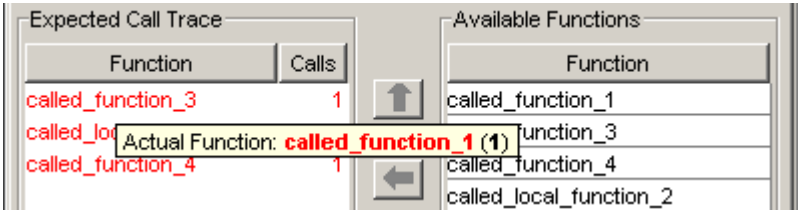
In this case, the tooltip indicates that function *called_function_4* should have been called, but **no** functions was actually called.

Example: Actual number of function calls wrong



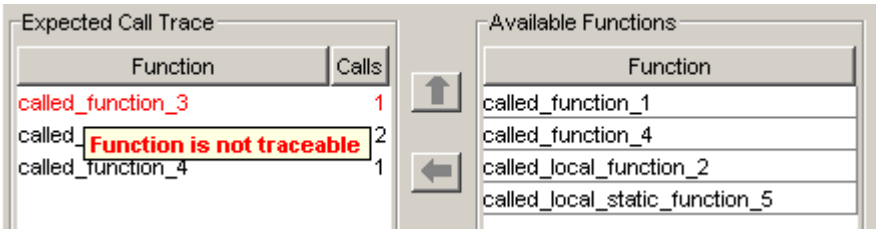
In this case, the tooltip indicates that the function *called_local_static_function_5* was expected to be called twice, but the function was actually called only once.

Example: Wrong function was called



In this case, the tooltip indicates that the function *called_function_1* was actually called, but another function was expected to be called.

Example: Function is not traceable

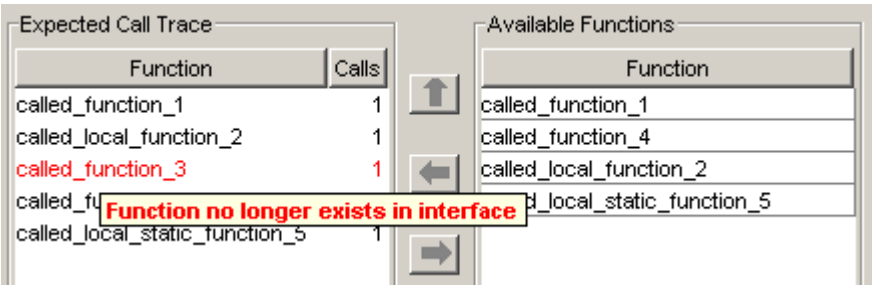


In this case, the tooltip indicates that the function *called_function_3* cannot be traced any more. This may occur due to changed settings within TIE: If an external function was

defined to be stubbed before (by activating the red tick), this function was member of the **Available Functions** list and could be used in the **Expected Call Trace**. Now, this function shall not be stubbed any more (e.g. because the function will be linked as object file) and though do not appear within the **Available Functions**, but it is still in the expected call trace.

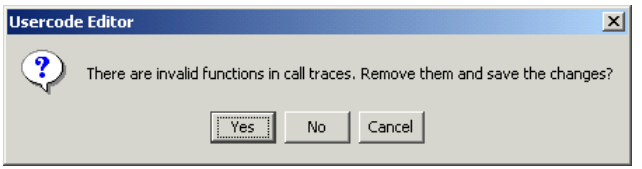
You may either remove the function from the **Expected Call Trace** or change the settings back within TIE to create a stub function for this function again.

Example: Function no longer exists in the interface



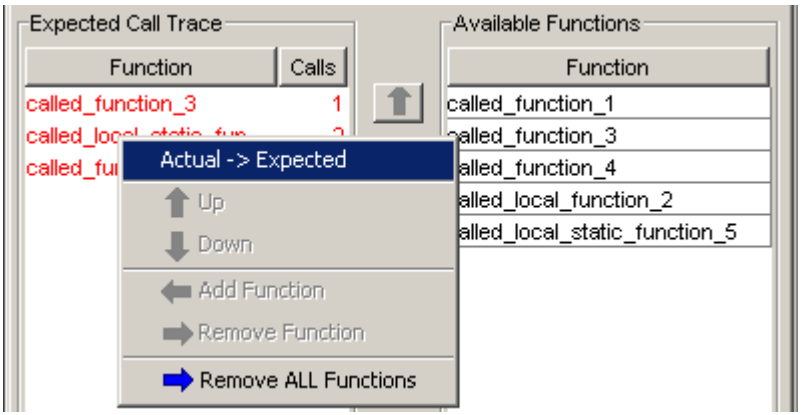
In this case, the tooltip indicates that the function *called_function_3* does not exist any more in the test object interface. This may occur due to changes within the source code, if the function is no more called from the test object or from other functions called by the test object. You need to remove the function from the **Expected Call Trace** and you probably also need to verify the resulting call trace, if there are more changes necessary due to the changes within your test object.

In the last two example cases, the following dialog will appear if you close the UCE:



- Select **Yes** to remove all functions from all expected call traces that are not more traceable or not more available within the interface of your test object. Use this option, if you know that there are only these changes necessary.
- Select **No**, if you want to save the Usercode as is without modifications to the expected call traces. You may remove all invalid functions later by opening and closing the UCE (this dialog will be shown again).
- Choose **Cancel** to do nothing and leave the UCE open.

If the actual call trace is as you wanted it to be, but your **Expected Call Trace** is something else (e.g. marked as wrong like in the above cases), you may use the actual call trace as the new expected call trace by using the context menu Actual -> Expected like shown below.



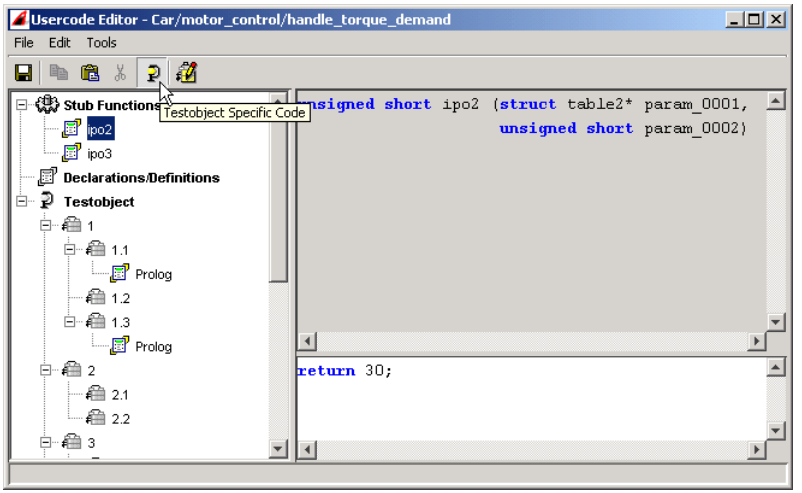
This overwrites all entries in the **Expected Call Trace** list and replaces the list with the actual call trace from the last test execution.

Test Object Specific Code

On the menu bar you will find an icon for **Testobject specific code**.

- When **Testobject specific Code** is activated, the currently edited Usercode section (e.g. declarations, definitions or the code of a stub function) is valid only for the current test object and not visible for the other test objects of the module. It is thus possible to create other definitions or stub function code for another test object

in the same module.



Compile Usercode

Before closing UCE and also in between changes to the Usercode, you should make sure, that the Usercode is syntactically correct by using the **Compile** button in the toolbar. Results can be seen within the message window of Tessy.

Delete Usercode

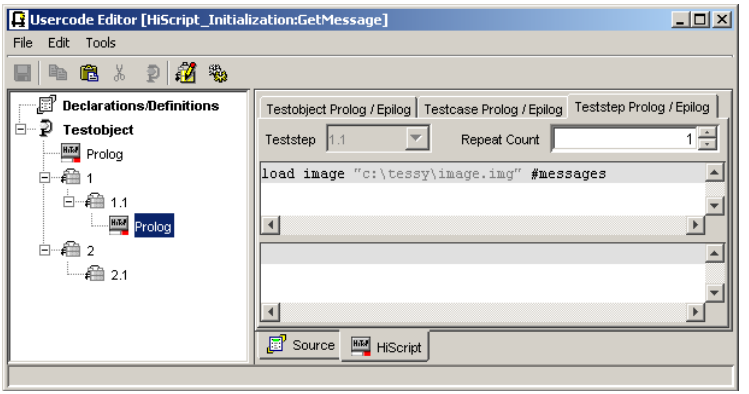
You can delete the Usercode in the current editor pane by choosing **Select all** in the **Edit** menu or by pressing `[Ctrl]+[A]`and pressing the `[Del]` key afterwards.

- If you want to delete the whole usercode of a test object, choose **Delete all Usercode** in the **Edit** menu.
- If you have deleted all Usercode by mistake close UCE without saving to cancel all modifications.

Enter Script Commands

The Usercode supports the native scripting language of certain emulators for use in prolog and epilog sections. Depending on your **Target Settings** for the module (refer to Module Properties) an additional tab within the Usercode editor will be inserted, e.g. **HiScript** for use with HiTOP emulators.

Please refer to the application note **Using Usercode** for details. Tessy currently supports scripting for TRACE32 (PRACTICE commands) and HiTOP (HiScript commands). You may add any command of the scripting language into the prolog and epilog sections within the script tab.



In the example above, a previously saved data image is loaded to initialize the *messages* variable. The scripting language depends on the emulator in use, please refer to the manual of the respective emulator for details.

Close Usercode Window

When you have entered the Usercode, click **OK**. The Usercode will be saved into the test database. If you do not want to save the changes, click **Cancel**.

Import/Export

Tessy provides an interface to store all test data of a test object into one file. You can edit the exported test data and import them back into the test database of Tessy. This may be

useful when editing a large number of test cases with only a few differences. You may choose different export formats like **.txt** or **.xls**.

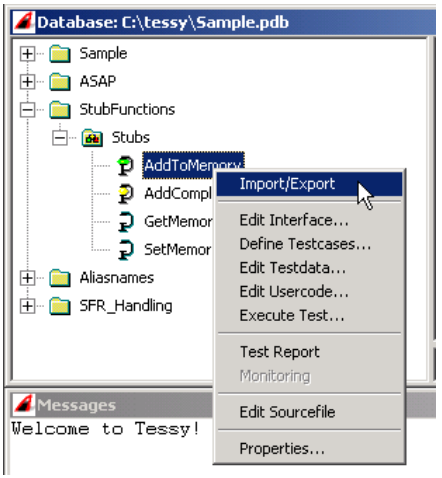
It is sometimes more convenient to enter the test data into an Excel sheet and then just import them into Tessy. In this case, you should create a dummy test case, initialize the values to zero (using the init feature within TDE), and export this test case.

You will end up with a perfect template file for this test object and you just need to fill in the input and expected values into the values sheet of the Excel file.

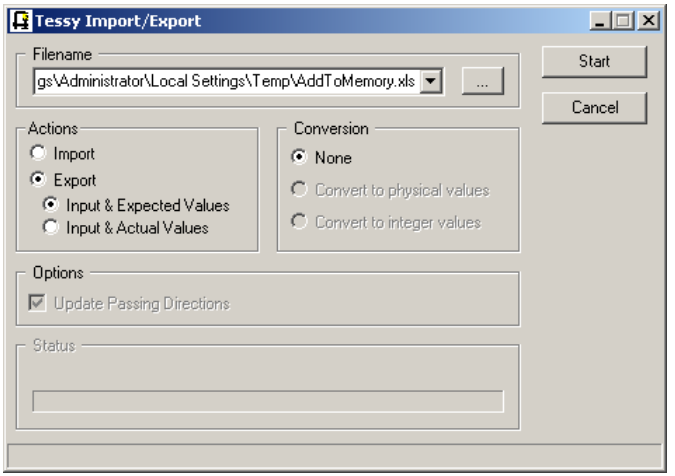
Important note: *You have first to initialize all test cases to be able to export test data.*

To export or import test data of a test object, do the following:

- Select one test object and one or more test cases from the **Unit** tab. In case that there is no test case selected, Tessy will export all test cases.
-
  Click either the Import/Export icon in the toolbar or choose **Import/Export** from the test object's context menu.



The Import/Export dialog will open.



- Tessy will use the test object name as file name by default. You may change this name directly in the upper text box or use the browse button to specify the file. You may use `*.txt` or `*.xls` to assign the file format.
- **Export Option:**
  To export test data select **Export** from the **Actions** section and choose **Input & Expected Values** or **Input & Actual Values**. Both options are only available, if a test run has been successfully executed. Otherwise, you may only export the input and expected values (default).

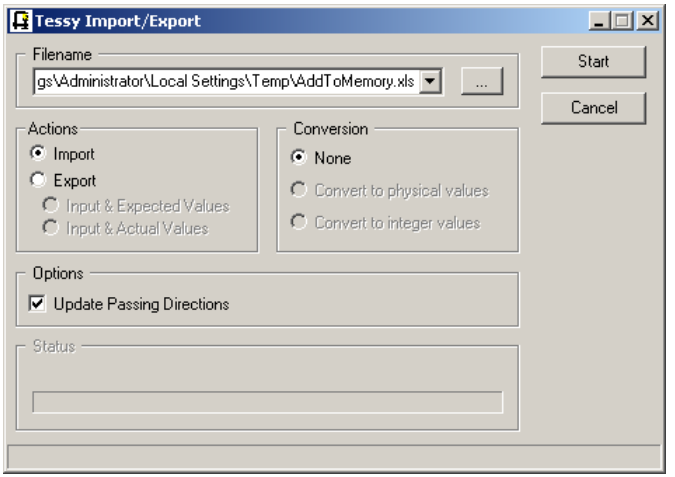
Note: *Storing the actual results may be useful, if you want to re-import these values as expected results, if the actual values are correct. In this case please make sure, that the actual values are REALLY what you expected! Otherwise, future test runs would check against wrong results.*

- **Import Option:**
  To import test data select **Import** from the **Actions** section and tick **Update Passing Directions** if necessary.

Note: *Check the box, if you want Tessy to change the interface settings. As a result, the passing direction of every variable that will be assigned a value during import will be set to In, Out or InOut, depending on the data within the import file. The passing directions of all other variables of the test object interface (not contained in the import file) will be reset to Irrelevant.*

- **Conversion Option:**
  These settings apply only if you have specified a conversion method within the module properties dialog (s. chapter Module Properties :: *General Tab*). Otherwise the default setting will be **None**.

  Please refer to application notes **Using ASAP Information** from the **Help|Documents** menu. You may choose to convert the imported data into physical values, if your import file contains the integer representation of the values. In this case, the conversion method will be used to convert all values to physical values to be stored within the test database.



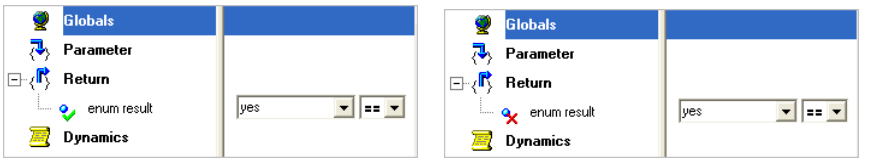
- Click **Start** to import or export test data. The status bar will show you the progress of the operation.

Enter Test Data

The Test Data Editor TDE

Upon creating of test cases, you have to enter test data using the Test Data Editor **TDE** to specify the **Input** and **Expected** values.

Input values of a test object are needed for the test execution of the given test object. The expected values are the calculated results of the tester regarding the input values for the test object after test execution. Tessy will compare both expected and actual values after test execution. The result will be either failed ✗ or passed ✓.

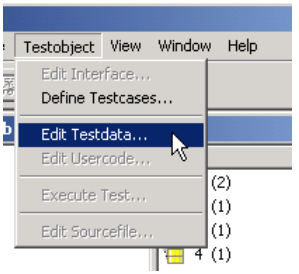


The decision to specify parameters and global variables as input values (`In`) or expected values (`Out`) has already taken place during test preparation with the **TIE** (see The Test Interface Editor).

Start the TDE

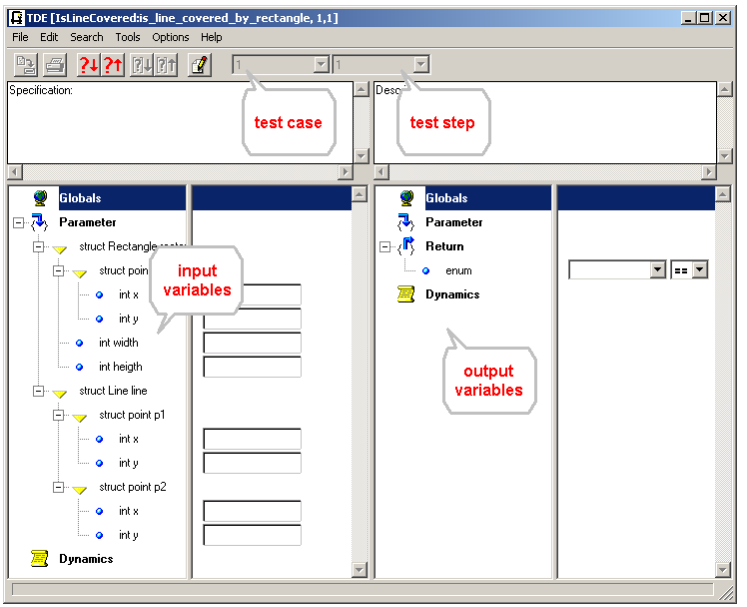
There are several ways to start the TDE:

- Select a test case or test object and choose **Edit Testdata…** from the context menu.
- Choose **Edit Testdata…** from the **Testobject** menu



- Click on the icon **Edit Testdata** in the toolbar.

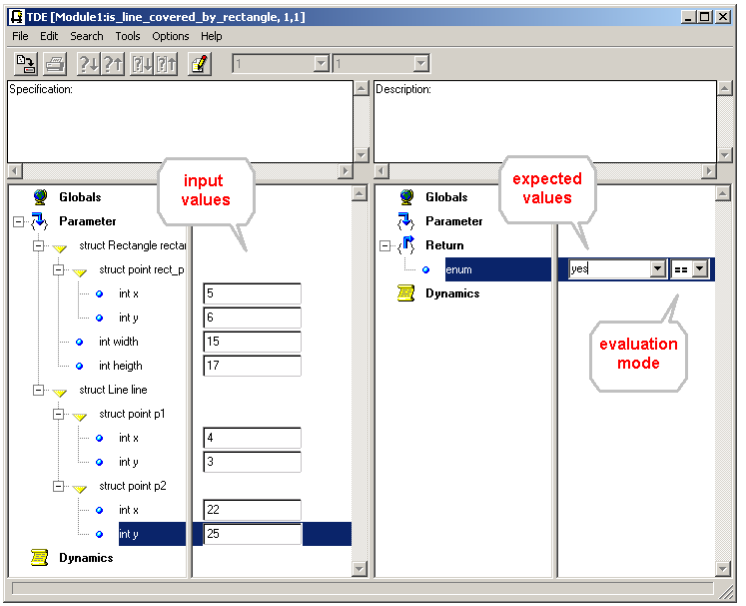
- Or double click the test case for which you want to enter test data.
  The TDE will open.



Structure of the TDE

The TDE is divided into two parts:

- The upper part with two text windows displays the test case specification and the optional test case description (s. Test Case Specification and Description).
- The lower part contains two windows, each of them divided into two panes. The right pane of the left window is used to specify the input values of the test object. The right pane of the right window specifies the output respectively the expected values and the evaluation mode.

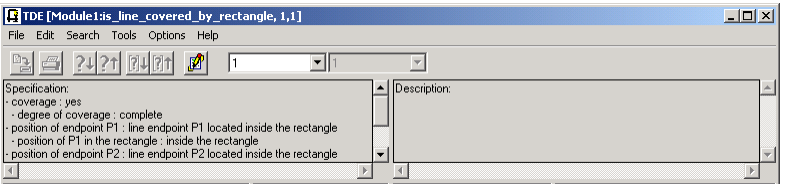


Test Case Specification and Description

The upper part of the TDE displays the test case specification and the optional description of the current test case in individual input fields.

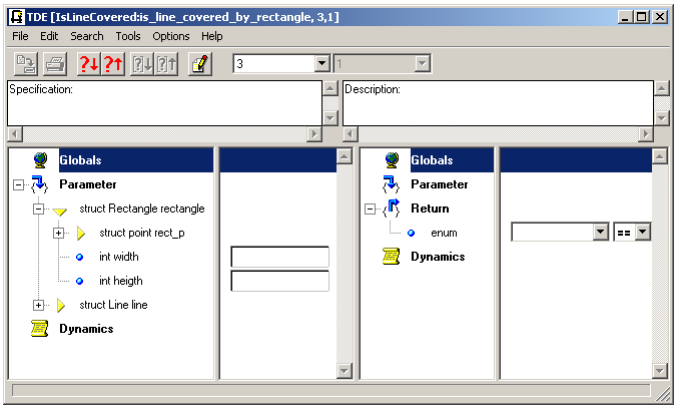
The actual test case specification should a tester enable to provide concrete input values and expected results for this test case. The description is optional and may be empty.

***Note**: The description and specification panes are not editable if test cases have been created and exported by using the CTE. The respective fields are displayed then in gray (s. picture below).*


TDE [Module1:is_line_covered_by_rectangle, 1,1]
File Edit Search Tools Options Help
Specification:
· coverage : yes
· degree of coverage : complete
· position of endpoint P1 : line endpoint P1 located inside the rectangle
· position of P1 in the rectangle : inside the rectangle
· position of endpoint P2 : line endpoint P2 located inside the rectangle
Description:

Browse the Interface

The TDE shows the interface of the test object which has been adjusted during the test preparation using **TIE** (s. Test Preparation: Set Passing Directions).



The interface elements are structured into the following sections:

Global Variables	Variables referenced by the test object (either locally defined or external). This section also contains global static variables and static local variables defined within functions.

Parameter	Parameters of the test object
Return	Return value of the test object
Dynamics	Represent pointer targets, referenced through a pointer of the test object interface (see Pointers)

Each section node holds its respective interface elements as child nodes (if any). You can open a section to browse through the interface tree.

***Note**: You may expand/collapse all items of a subtree if you right-click a section or one other node in the tree.*

The browser for input values and expected values is vertically divided into two parts for each set of values:

- The left window is used to specify the input values of the test object.
- The right window specifies the output respectively the expected values and the evaluation mode.

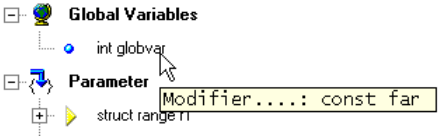
You will find a structure icon in front of every element when you open a section node of a browser. The icons are explained in the following table. The elements of different levels can be recognized in the browser by their differing indentations.

Icon	Description
	This icon indicates that a further level exists. The interface element in question is of a complex C type.
	This icon represents the browsed open state of a complex C type (*struct, union, array*). The sub elements of this type are represented below and indented by one level.
	This icon indicates a basic type that cannot be browsed open.

View Type Modifier

Type modifier, such as far, near, const (etc.) can be displayed within the TDE as follows:

- Press [Shift] and move the mouse pointer over a variable. *The type modifier will be displayed as tool tip (if any).*

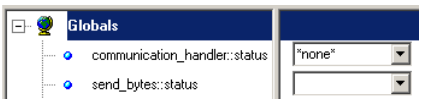


Special Global Variables

As mentioned above, the **Global Variables** section contains variables with the following lexical scope:

- Defined within the source file
- Declared within the source or header files
- Defined **static** within the source file
- Locally defined **static** within the **scope of a function** or at the beginning of a statement block

In case of the first three bullets, there will be no difference in the display of the variables. In the last case of static local variables, the name of the variable will be the function name followed by double colons and the variable name (in order to distinguish identically named local variables within different functions).

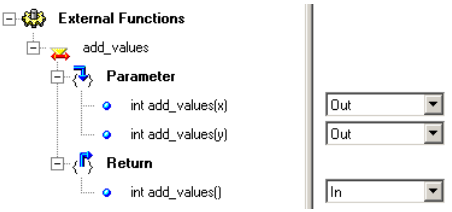


If there are nested static local variable definitions within the same function, the names will be enumerated automatically.

***Note**: Some restrictions apply when setting input values for static local variables. Refer to " Input Values for static local Variables" for more information.*

Advanced Stub Variables

The parameters and return values of advanced stub functions which have been defined in TIE (s. next picture) will also appear within the **Global Variables** section of TDE.



The following naming convention will apply for these variables:

- The parameter variables will be named using the stub function name followed by the name of the parameter in parenthesis.
- The return value will be named using the stub function name followed by parenthesis.

Example:

```
//
// External function to be called
//
extern int add_values(int x, int y);
```

For the external function shown above there will be three advanced stub variables available, two for the parameters and one for the return value (if not set to `Irrelevant` within TIE).

Within a test step, you may check the values passed to the parameters (they are `Out` variables for this reason) and you may provide a value to be returned by the stub function (therefore the return value is an `In` variable).

In the example above, the parameters x and y are checked for the values 2 and 3 and an (erroneous) return value 13 is provided instead of the correct calculation which would be 5.

Shortcuts for Navigation

Following shortcuts may be used to navigate through the TDE (s. also Copy and Paste Test Data).

[Tab]	will step to the next variable.
[Shift]+[Tab]	will step backward to the previous variable.
[Ctrl]+[Tab]	will jump to the input pane or output pane.
[Ctrl]+[Page Down]	will display the next test step of a test case
[Ctrl]+[Page Up]	will display the previous test step of a test case.

Test Steps

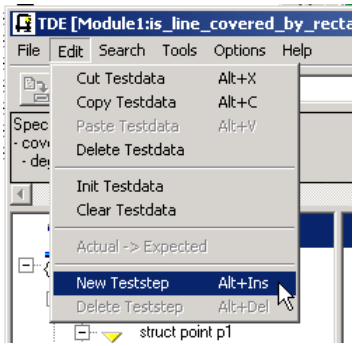
You can add additional test steps to each test case. Every test step contains a complete set of test data.

For instance, the mechanism of test steps can be used to achieve an initialization of the test object before executing the test step that checks the actual test condition of the current test case

Create Test Steps

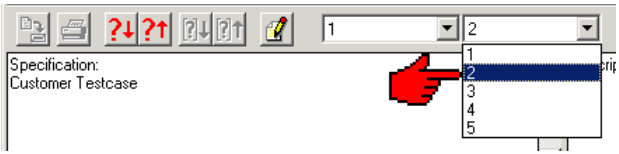
To insert additional test steps, do the following:

- Either choose **New Teststep** from the **Edit** menu or use [ALT]+[Ins].



The TDE will automatically show the newly generated test step.

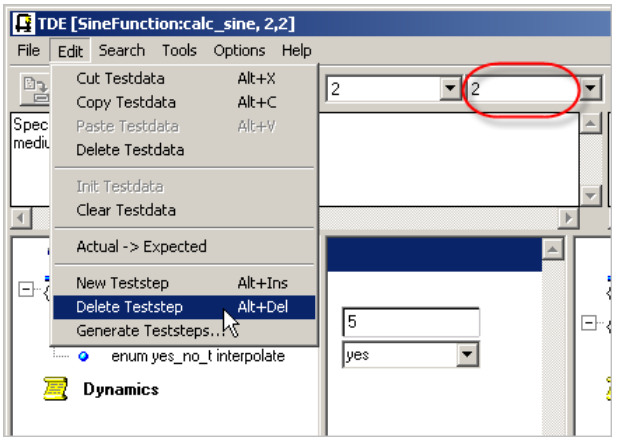
If there is more than one test step in your test case, you may choose the appropriate one using the respective combo box.



Delete Test Steps

To delete a test step, do the following:

- Choose the test step which you want to delete, e.g **2**.
- Select **Delete Teststep** from the **Edit** menu or use **[Alt]+[Del]**.



TDE will delete the test step and the corresponding test data and dynamic objects.

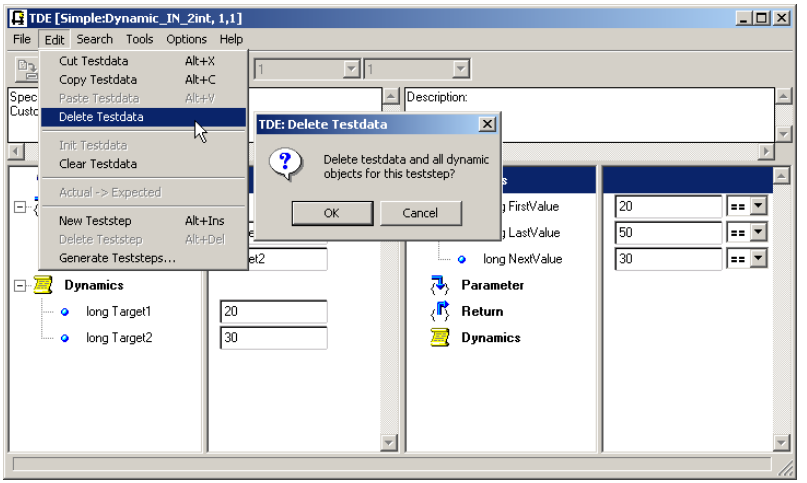
***Note**: By default, at least one test step will be left and cannot be deleted (s. Delete or Clear Test Data for a Test Step).*

Delete or Clear Test Data for a Test Step

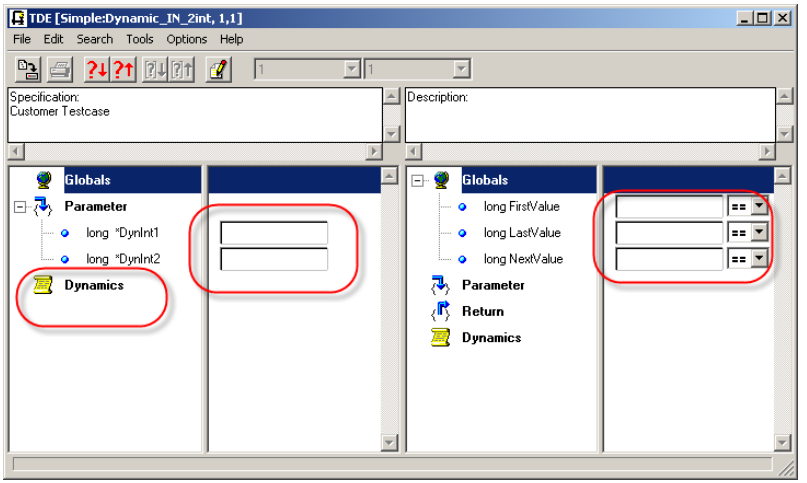
You may remove the test data for a test step by using either option **Delete** or **Clear**.

Delete will completely remove the test data and dynamic objects (target values) of the test step.

- Choose **Delete Testdata** from the **Edit** menu.

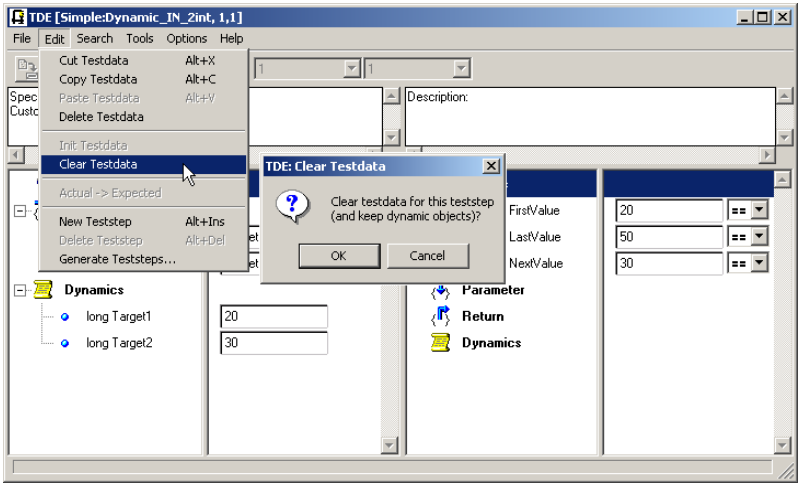


If you click OK, the test data and the dynamic objects will be removed.

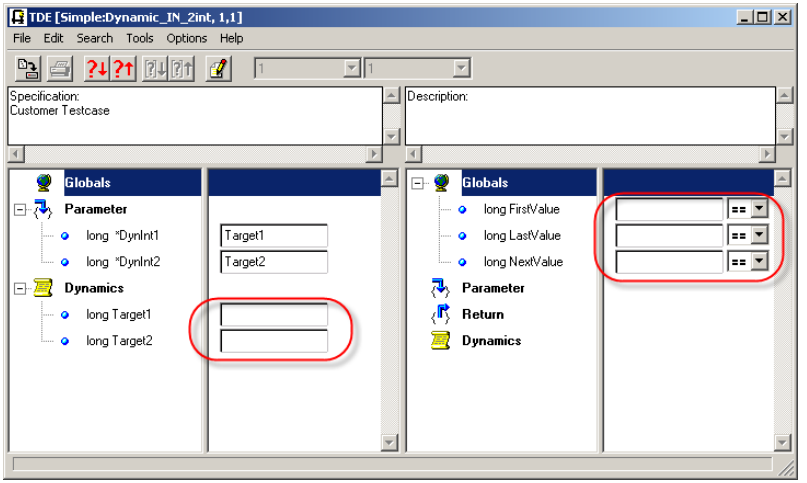


Clear will only remove the test data of a test step but will keep dynamic objects (target values).

- Choose **Clear Testdata** from the **Edit** menu.



If you click OK, only the test data has been removed but dynamic objects will be kept.



Specify Execution of Test Steps

Tessy provides the feature to execute test steps repeatedly. The number of repetitions can be specified in the user code prolog of a test step (see Repeat Test Step Prolog/Epilog Execution).

- You can edit user code directly using **Edit Usercode…** from the **Tools** menu or click the respective icon in the toolbar.

Tessy will use the same input values for each repetition. Only the actual values of the last repeat cycle will be stored to the database.

Edit Input and Expected Values

The interface settings carried out within TIE determine if a variable will be displayed within TDE and if you may enter an input or expected value or both values.

***Note**: All interface elements with the passing direction not set to "Irrelevant" or "Extern" will be displayed within the TDE.*

Every variable will be assigned to one of the section elements described above, e.g. **Parameter, Global** etc. Initially, the **Dynamics** section will always be empty (s. Browse the Interface on page 218).

Input fields for each interface variable will be displayed for input values and/or expected values depending on the used passing directions. The passing direction of dynamic objects is depending on the passing directions of the pointer variables pointing to the respective target object (s. chapter Test Preparation :: Pointers).

Input Values

Input values are all interface elements or dynamic objects that need to have a defined value at the beginning of the execution of a test object. These values are included in the calculation of the output values during test execution, or will be used as a predicate of a condition.

There are three types of input values:

- **Global and external variables**, used by the test object.
- **Function parameters**, transferred during function call.
- **Dynamic Objects**, they represent pointer targets, referenced through a pointer of the test object interface (they are not really dynamic, variables will be created for each one within the generated test driver)

Input Values for static local Variables

Static local variables require special handling within the generated test driver, causing some restrictions for the data input of the test cases.

Example:

```
void send_bytes()
{
   // The internal status of the state machine
   static state_t    status = STATE_IDLE;
```

Normally, static local variables are not accessible outside the scope of the function or statement block where they are defined. To be able to assign input values and save the actual results of the expected values, Tessy adds some code after the definition of the static local variable. This code initializes a pointer to the static local variable. This pointer will then be used for each subsequent access to the variable within the test driver.

As a consequence, this pointer needs to be initialized **before** Tessy may assign input values to the static local variable. The initialization takes place upon the first execution of the code Tessy added after the definition of the variable.

***Note**: Input values for static local variables may not be assigned until the first execution of the definition scope of the variable (e.g. the first execution of the test object, if the variable is static local within the function body of the test object). Static local variables cannot be used in the UCE.*

For a test object containing static local input variables, the first test step would always require to ignore the input variable (causing the initialization value of the variable within the source code to be used).

Any further test step could then be used to assign arbitrary values to this variable (assuming that the definition scope of the variable will be executed by the first test step).

***Note**: Beware of this restriction when debugging single test cases. The input settings for static local variables of at least the first test step to be executed will be ignored because of this reason.*

Expected Values

The expected values determine what results are expected after test execution.

Expected values can only be defined for interface elements or dynamic objects that have a defined value after the execution of a test object.

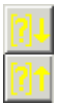
There are the following types of expected values:

- **Global or external variables** can always be expected values as they have a valid value after execution of a test object.
- The **Return Value** of a function is always an expected value.
- **Dynamic Objects**, referenced through a pointer that is either an input or expected value, continue to be accessible after execution of a test object. They can therefore also be expected values, just like global variables.

Enter Values

For entering values for the interface elements, suitable input fields are provided in the TDE. Initially, all values are undefined. When running the test with undefined values, the initial value passed to the test object depends on the default initialization of the compiler used.

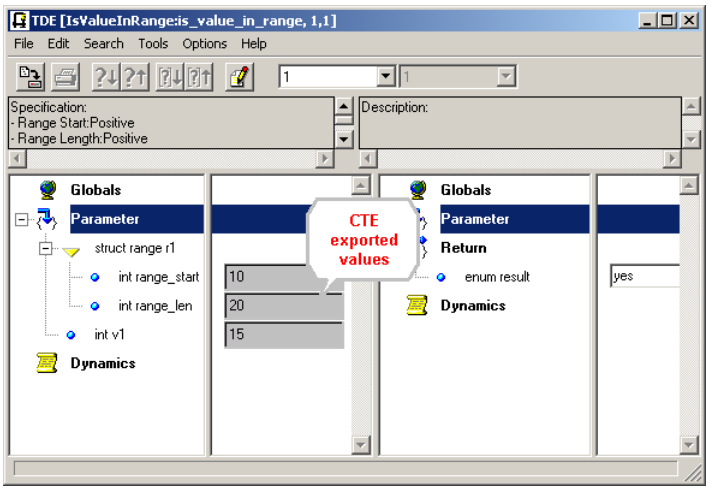
If at least one variable in the current test step is undefined, the icons **Next Undefined Variable** and **Previous Undefined Variable** will be activated. If you click on these icons, the cursor will automatically jump to the next undefined variable.



The icons **Next undefined array element** and **Previous undefined array element** work similarly for arrays. Arrays do not necessarily be completely initialized, providing values for one array element is sufficient.

CTE Exported Values

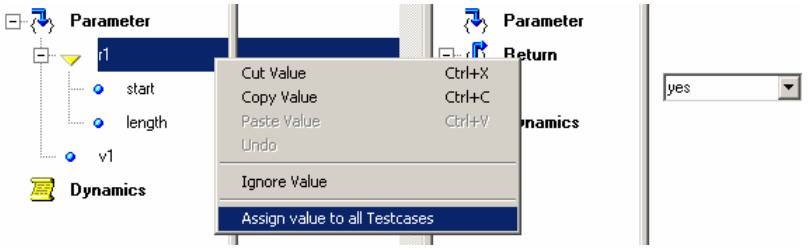
CTE exported values for variables are read-only in TDE. The respective input fields are displayed in gray.



You must use CTE respectively the underlying CTE document to change such values if necessary.

Assigning Values for Variables in all Test Cases

For convenient editing of new interface values or even for initialization purposes during test data input, you may assign the current value of a variable to this variable within **all** test cases .



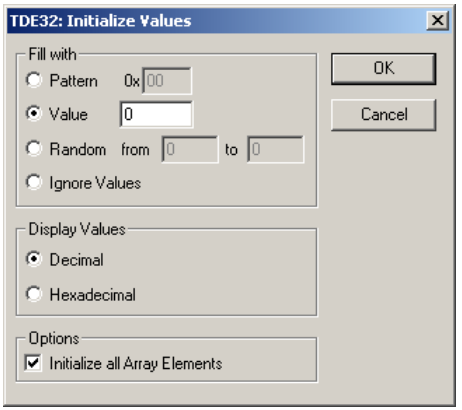
This menu is available as soon as the respective variable has a value for the current test case. Even structs, unions or arrays may be assigned to all test cases (and test steps) in this way.

Initialize Test Data At Once

TDE provides an option dialog to initialize all interface variables of a test step at once.

To initialize variables, do the following:

- Choose **Init Testdata** from the **Edit** menu.
  The following dialog will open.

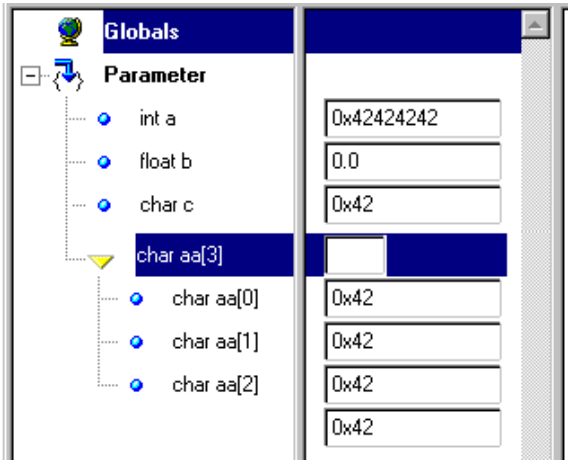


You may use following options:

- **Pattern**
  a pattern in hexadecimal format.
- **Value**
  a specific value only.
- **Ignore Values**
  all input and expected values will be set on ***none***.
- **Display Values**
  you may decide to display **Value / Random** in decimal or hexadecimal format.
- **Random**
  a range of generated values for the initialization. The random values will adhere to the min/max limits of each interface variable type.

- **Initialize all Array Elements**
  if selected, all array elements will be initialized. Otherwise only visible array elements will be initialized.

In accordance with the size of the type of a given variable, an initialization value is derived from the pattern in hexadecimal notation (e.g. if you entered the pattern 0x01, the initialization for a 16 bit value would be 0x0101).



All interface components are treated according to their type. Dynamic objects are also initialized.

The following table shows the initialization values for certain data types.

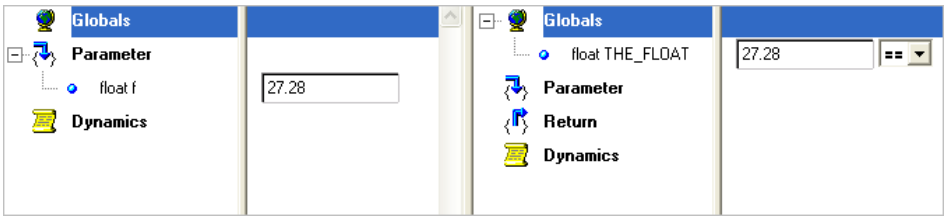
Type	Contents
`Integer`	`0x00000000` e.g. if `0x42` is entered as initialization value, all `int` variables will be initialized with `0x42424242`.
`Float`	`0.0`
`Char`	`0x00` or what is entered.
`Struct`	All sub components are initialized according to their type.
`Union`	The first sub component is initialized as active component.

Enum	The first **enum** constant is used as initialization value.
Array	All array elements are initialized according to their type (if option "**Initialize all Array Elements**" is used)
Pointers	Pointers are initialized with "**NULL**" provided that they don't point to dynamic objects.

***Note**: If values have already been entered for some components it is still possible to initialize the remaining components. In this case only the undefined components will be initialized.*

Enter Numerical Values

Integers and floating point numbers are entered into a simple text box, e.g. **27.28**.



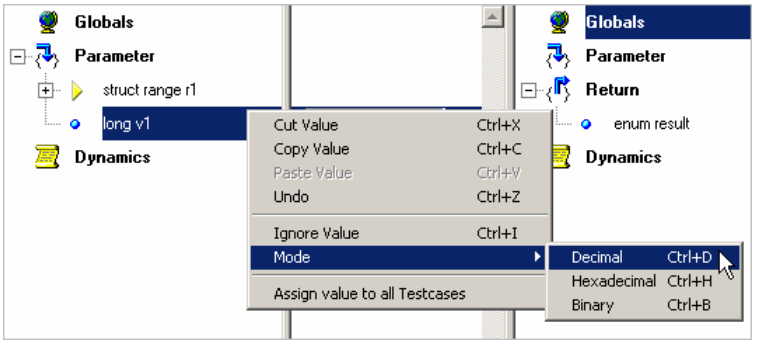
***Note**: You may select **Enable checking** from the **Options** menu for strong syntax checking to prevent inappropriate values for a given data type.*

Enter Values in Decimal, Hexadecimal or Binary Format

You may provide the input or expected values in three different formats:

```
0b00000001     :: binary
0x01           :: hexadecimal
1              :: decimal
```

You may either use the context menu of the variable switching to the appropriate mode or you may simply type `0x` or `0b` in the input field to use hexadecimal respectively binary format (s. “Bitcheck”).

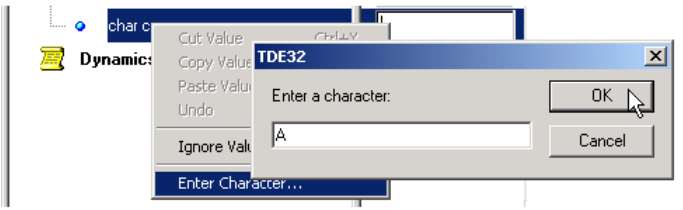


In this way, you may also convert a value in decimal, binary or hexadecimal format by using the context menu or the appropriate short-cuts.

Enter Characters

For a single `char` variable you may also use the special Option **Enter Character**… from the context menu.

- Select the variable name and choose **Enter character** from the context menu. *The following dialog will open.*



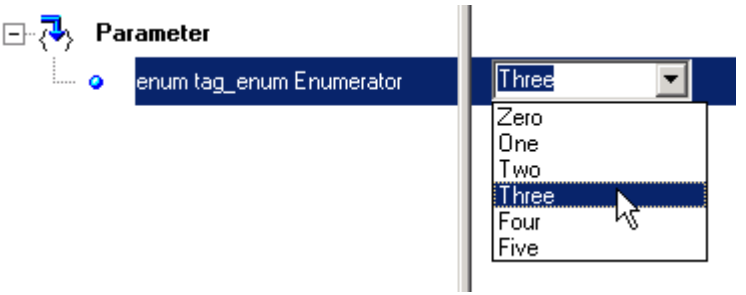
- Enter a character, e.g. `A` and click **OK**.

The **TDE** will enter the ASCII code of the character as hexadecimal value into the respective box. (s. Enter Strings)

Enumeration Types

For the input values of enumeration types, e.g. `enum` variables, you may select one of the defined enumeration constants from the combo box.

You may also directly enter the name or a numerical value for the enumeration type.



Choose the name from the list...

...or enter a numerical value for the constant. This will cover the case if an undefined constant is given.

Edit Structures and Unions

Structures (*struct*) and unions (*union*) are processed as structured types in the **TDE**. If you browse open them, the components of the struct appear in the next structural level.

The values for the components are entered like normal interface elements of the according type.

There is no possibility to enter values for the structure itself. Only unions have active components in the combo-box. If there is more than one component available, the selected component will turn sensitive while the other components turn insensitive.

You can only enter values for the active component of Unions. All other components cannot be selected, because they are insensitive.

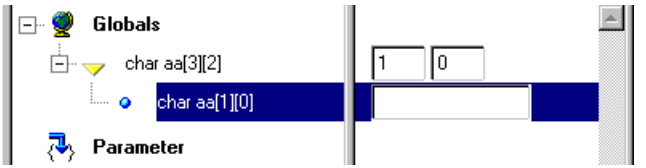
Arrays

Arrays will be displayed as a single entry with one or more input fields next to it. Each of these input fields represent one dimension of the array, e.g. you will see two input fields for a two dimensional array.

To enter data for a specific array element:

- Enter the indices of the array element into the input fields and press `[Return]`.

  The array element will now have a child entry for a specific array location.



- Enter an appropriate value.

Tip: *When you have defined a value for an array element, press* `[Ins]` *to return to the first array dimension field. Now, you can enter the indices for the next array element.*

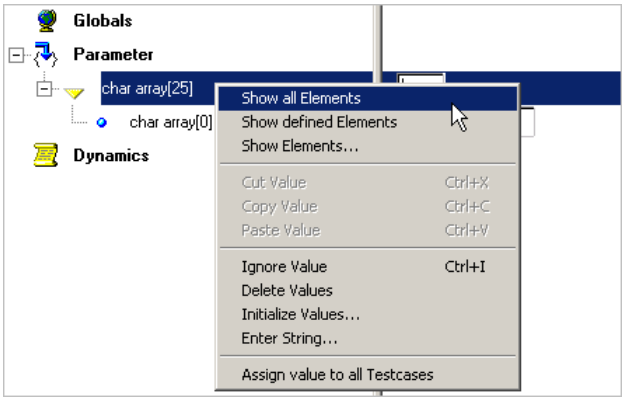
By default, **TDE** shows always the first array element even if it is empty. You can remove this setting using **Show first array element** from the **Options** menu.

***Note**: Choose **Save as Default** from the **Option** menu to preserve your setting.*

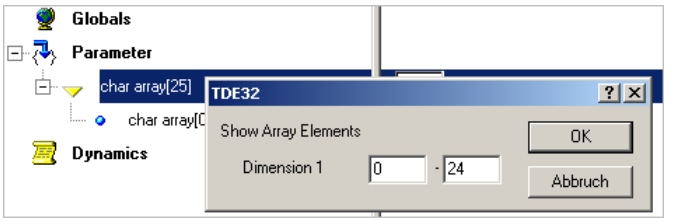
Show all or only defined array elements

If you want to display all array or only defined elements, do the following:

- Select the array name and choose either **Show all Elements** or **Show defined Elements** from the context menu.



- You may also use **Show Elements** to display a range of elements or one of the array elements.

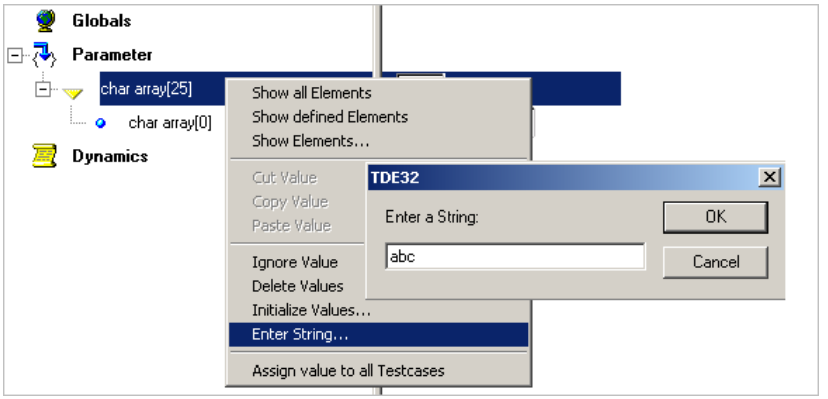


Enter Strings

Special forms of arrays are `char` arrays. You may enter a string instead of a value for each element separately. This will also work for signed and unsigned char arrays.

To enter a string, do the following:

- Select the name of a char array and choose **Enter string** from the context menu. *The following dialog will open.*



- Enter a string and click **OK**.

***Note**: The number of characters of the string is calculated by number of array elements – 1, because of the final \0.*

The TDE will enter all characters into the input fields of the array elements.

Pointers

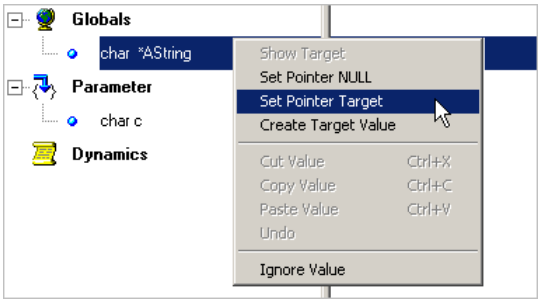
The values of pointers (i.e. target addresses) can only be entered directly for pointer to functions and void pointer.

In this case, you may enter a symbolic name or even an absolute address into the input field next to the pointer variable. For all other sorts of pointers, you need to use the context menu to assign a value.

- You may set the pointer to NULL, point it to a specific variable within the interface or create a new variable as the pointer's target. In the latter case, the TDE will create a new element within the dynamic object section.

To enter values for pointers, do the following:

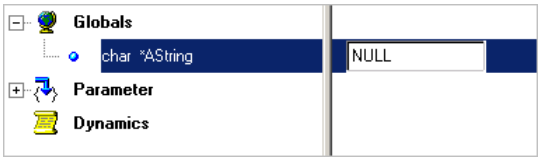
- Select the pointer name and open the context menu.



The context menu offers the following possibilities to assign a value for a pointer:

- **Set Pointer NULL**

  If you choose this option, the value of the selected pointer will be set to **NULL**. *The text box will be filled with NULL.*



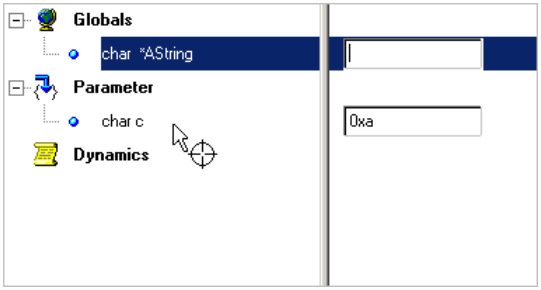
- **Set Pointer Target**

  If you choose **Set Pointer Target**, you can select another interface element (or a component of a structure or union) and assign its address to the pointer.

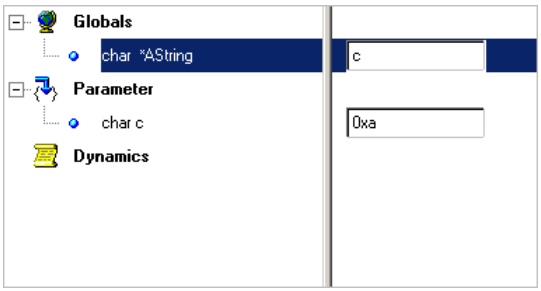
  The cursor will change, when you move the mouse pointer over a variable:

  The object type fits the pointers target type. You can assign the pointer.

  The object type doesn't match the pointers target type. You cannot assign the pointer.



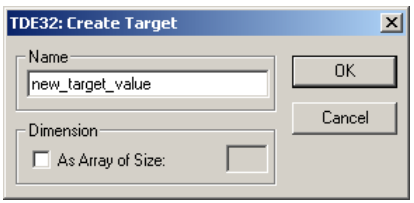
When you click on an element, the variable name of that element will be entered into the field next to the pointer. During test execution, the address of the variable will be assigned to the pointer.



- **Create Target Value**

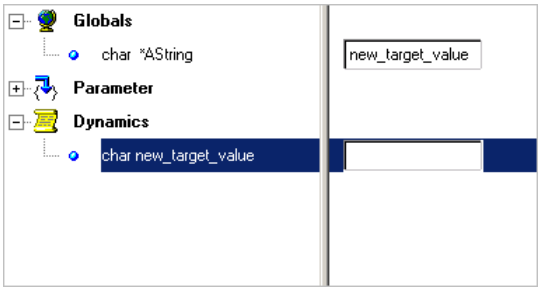
  Create Target Value allows to create a new object as target object for the pointer. The address of the object will be assigned to the pointer. The size and type of the created object depends on the target type of the pointer.

  Choose **Create Target Value** from the context menu to create a new object. *The Create Target dialog will open.*



Enter a valid C identifier as name for the new target object and click **OK.**

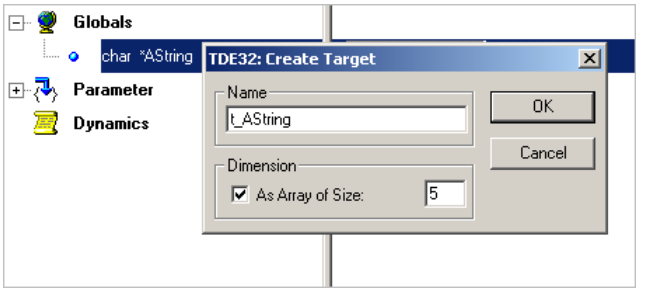
A new target object will be listed in the dynamic objects section of the TDE.



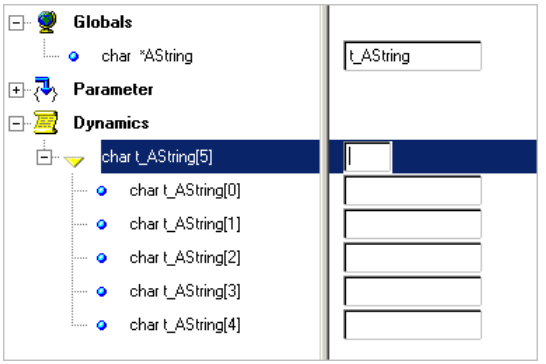
- **Array as Target Value**

It is also possible to create an array as target value using the **Dimension** option of the **Create Target** dialog.

You need to tick the check box for **As Array of Size** to enter an appropriate size into the input field. Click **OK**.



The name of the new object appears in the field next to the pointer. *TDE will create an array of the pointers target type. The pointer will point to the first array element.*



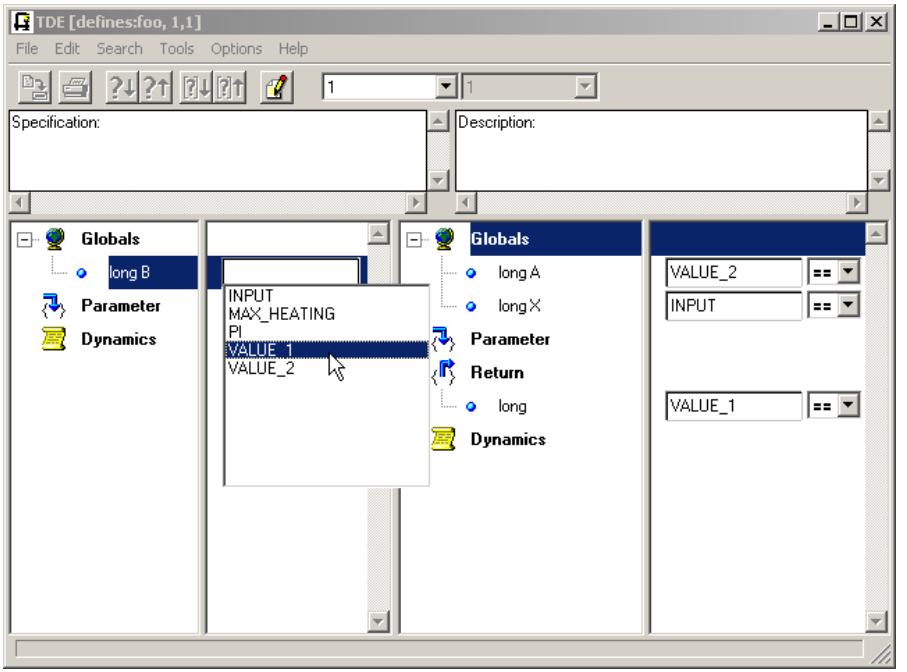
Within the **Dynamics** section, you will see the newly created target object. Next to it you can enter values, like for every other interface element.

Important note: *TDE doesn't support to create target values for void pointer. In this case you have to use the usercode editor* ***UCE*** *to provide an appropriate target value.*

Using Symbolic Constants

TDE will be able to use `#defines` of constant values as symbolic test data.

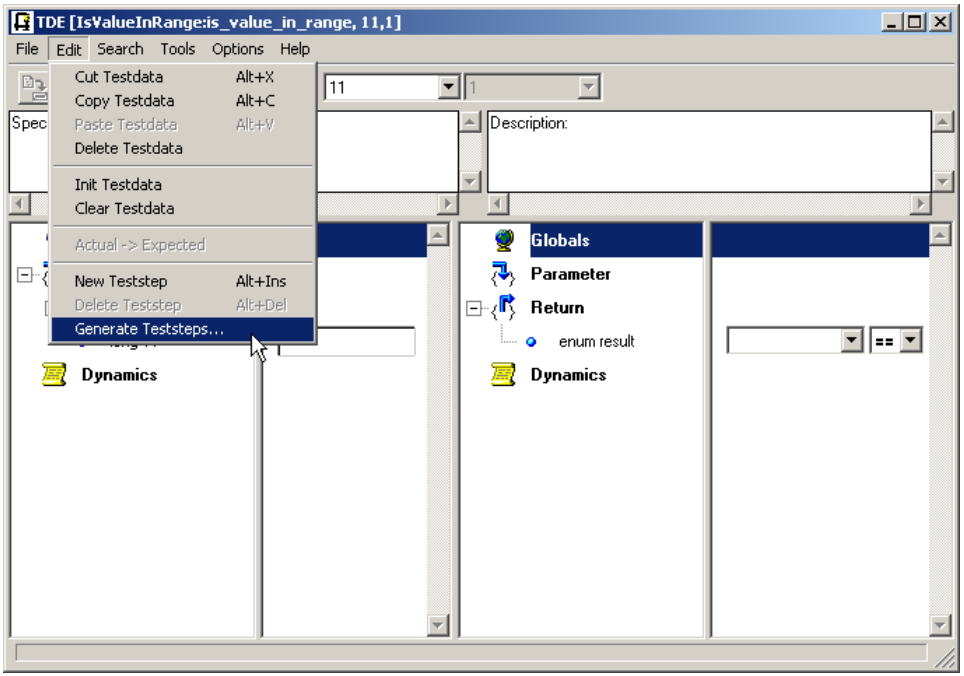
- Select the variable and press the `[Space]` bar.
- Choose one of the constants from the list or type the initial letter (and the following) of the constant in the field. This will direct you to the first hit in the list.
- Press `[Enter]` or double-click to assume the constant, e.g. `VALUE_1`.



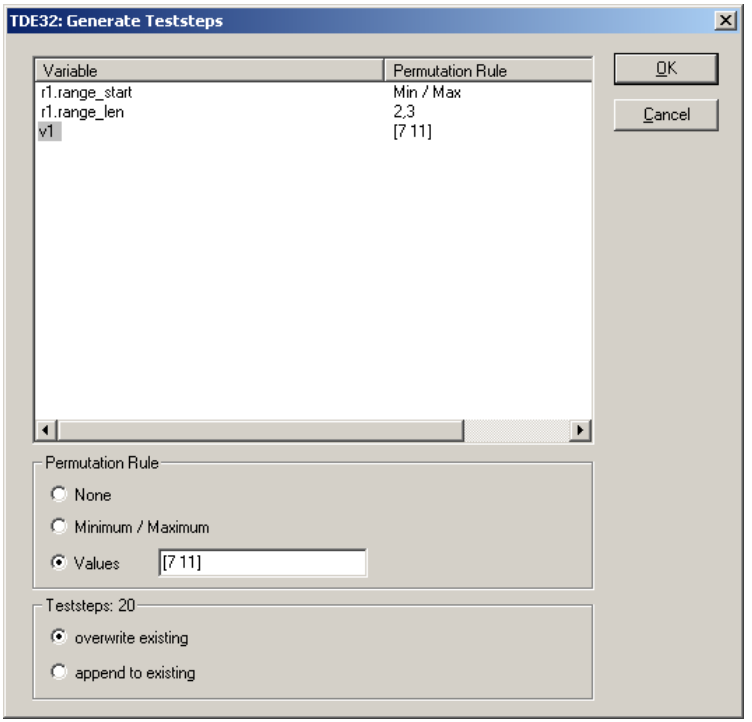
Automatic Test Data Generation

The **TDE** supports automatic generation of test data. The test data manifests in test steps of a single test case.

- Create a new test case (s. Edit Test Cases) or select an already created test case.
- Open **TDE**.
- Choose **Edit|Generate Teststeps…** from the menu bar.



The Generate Teststeps dialog will open



- **Permutation Rule**
  You may simply specify to use **Minimum** and **Maximum** values which are possible with the used data type. Alternatively, you may also specify a comma-separated list of values, e.g. `2,3` or a range of values by using squared brackets, e.g. `[7 11]`. The values must be separated by a blank. You may also combine both notations as follows: `2,[7 11],[19 22],25`

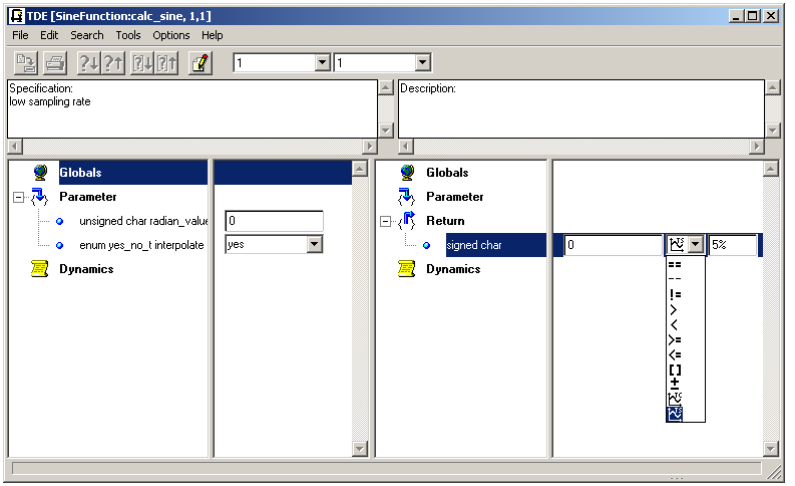
- **Teststeps**
  TDE calculates the number of test steps resulting from the current settings, e.g. `20`. This allows you to avoid inflation of test steps by adjusting the settings.

  You may append the generated test steps to existing test steps of this test case (**append to existing**) or you may create a test case consisting purely of generated test data (**overwrite existing**).

Enter Evaluation Modes

The right lower window of the TDE specifies the output variables respectively the expected values and the evaluation mode.

- For all variables enter the values you expect to be calculated during the test execution of that test step.
- Using the evaluation mode allows to specify how to compare the actual value (calculated during the test run) with your specified expected value. The default value is **==** (Equal).



The **Test Evaluation Mode** together with the **Expected Value** will be used to process the test results after the test run (see Test Report and View Results after Test Execution).

- To set an evaluation mode for a component choose an appropriate evaluation mode from the combo box. There are following standard evaluation modes available (s. Special Evaluation Modes for Result Plots):

  == (equal):
  checks the expected value and actual value for Equality (the default setting).

  != (unequal):
  checks the expected value and actual value for inequality.

  > (greater):

checks if the actual value is greater than the expected value.

< (less):
checks if the actual value is less than the expected value.

>= (greater or equal):
checks if the actual value is greater or equal to the expected value.

<= (less or equal):
checks if the actual value is less or equal to the expected value.

[] (range):
checks if the actual value is within a range.

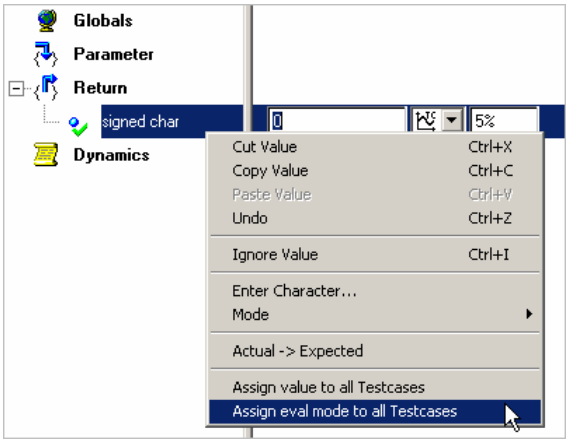
If **[]** is selected, TDE will display two input fields. You have to enter the boundaries of the range. The first value has to be the minimum, the second has to be the maximum value.

± (deviation):
checks the deviation of the actual result compared to the expected result.

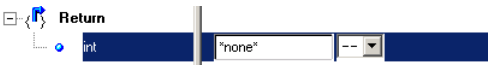
If ± is selected, TDE will display a separate input field. The deviation value may be provided as absolute number (e.g. `1`) or as percentage of the expected result (e.g. `1%`).

If the deviation value has a positive sign (e.g. `+2` or `+2%`), only deviations in the positive direction will be accepted, and deviations in the negative direction will not be accepted. If the deviation value has a negative sign (e.g. `-2` or `-2%`), only deviations in the negative direction will be accepted, and deviations in the positive direction will not be accepted. If the deviation value has no sign, deviations in both directions will be accepted.

If desired, the evaluation mode including the deviation value may be assigned to all test cases and test steps via the context menu **Assign eval mode to all Testcases**.



`-- (ignore):`
ignores the result. The actual value will not be processed after test execution and has no effect on the overall test result.

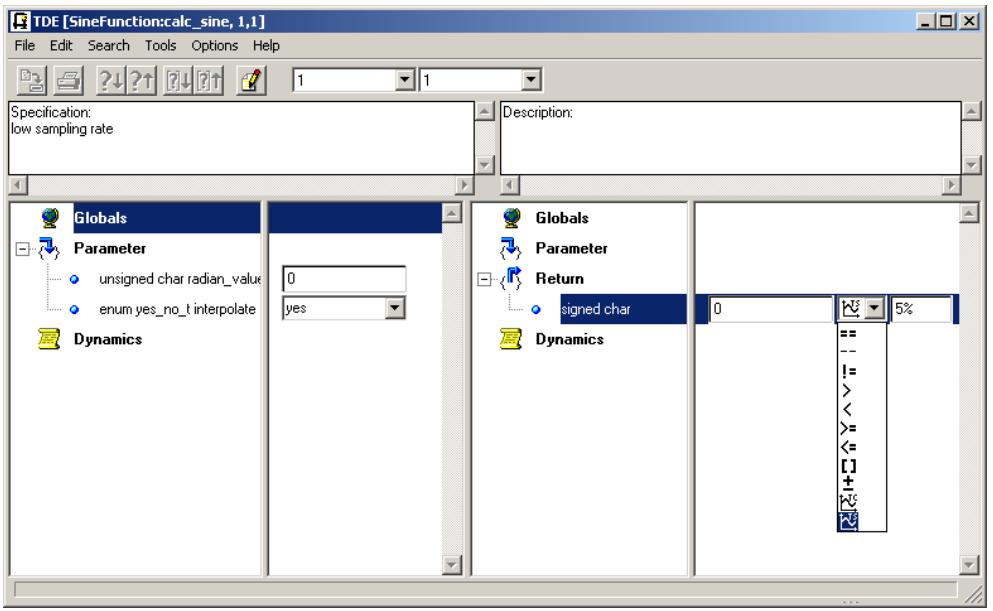


Special Evaluation Modes for Result Plots

TDE supports the graphical visualizing of test data for interface variables as signal data rows (s. chapter Test Report).

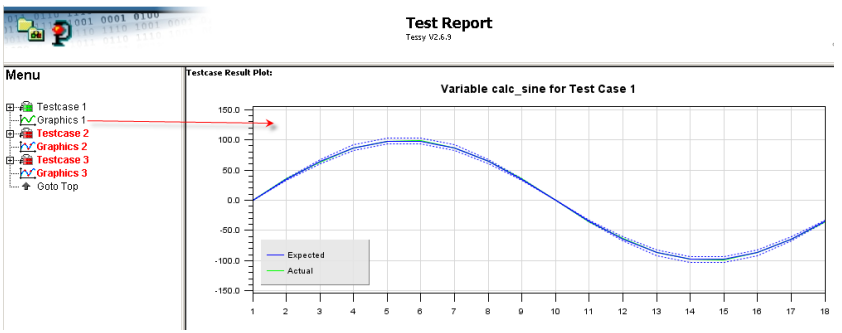
This will be available either for data rows within one test step (e.g. for arrays) as well as for data rows of a variable over all test steps.

Graphical result plots of interface variables will be available without additional software, e.g. MatLab.

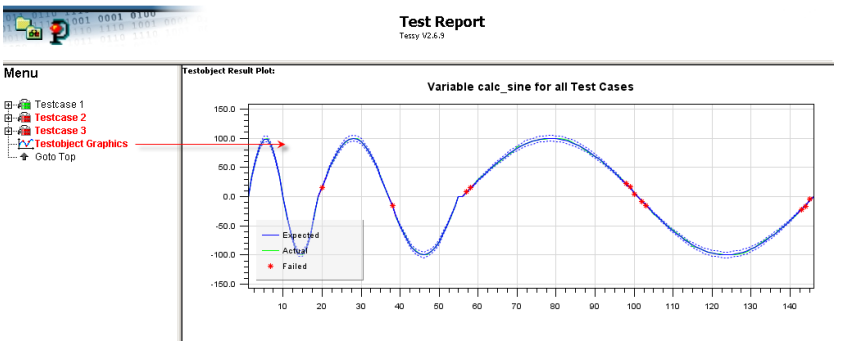


The appropriate mode for result plots can be used from the respective combo box in TDE. You may specify a deviation value for the evaluation in a separate input field, e.g. **5%**. The result plot is currently available for HTML Reports (s. chapter Test Report).

result plot of one variable over all test steps of each test case



result plot of one variable over all test cases

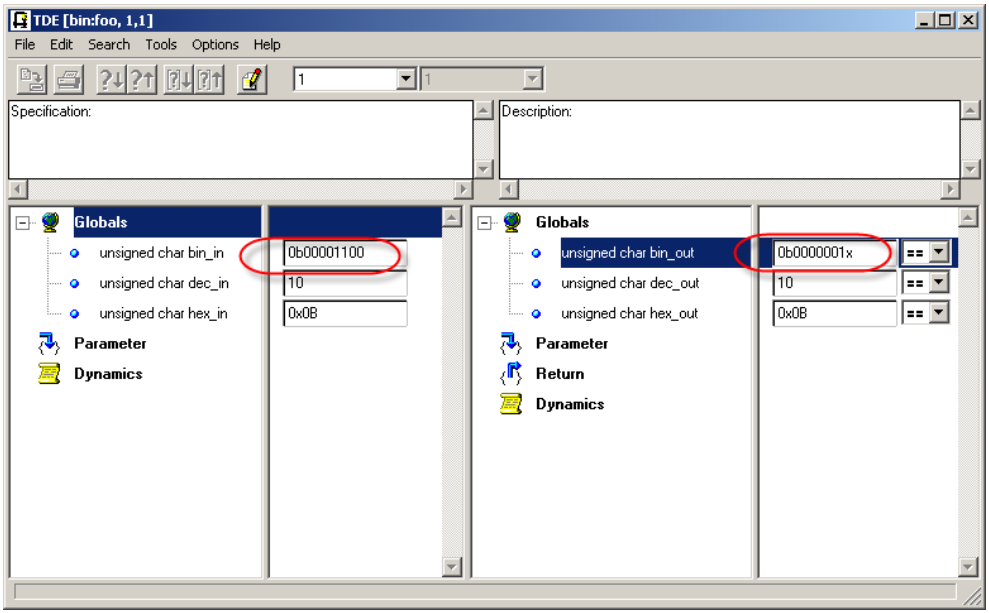


Bitcheck

Individual bits of a variable can be checked during the evaluation for a test case result. It will be possible to mask-out bits of an expected value in the **TDE**, e.g.

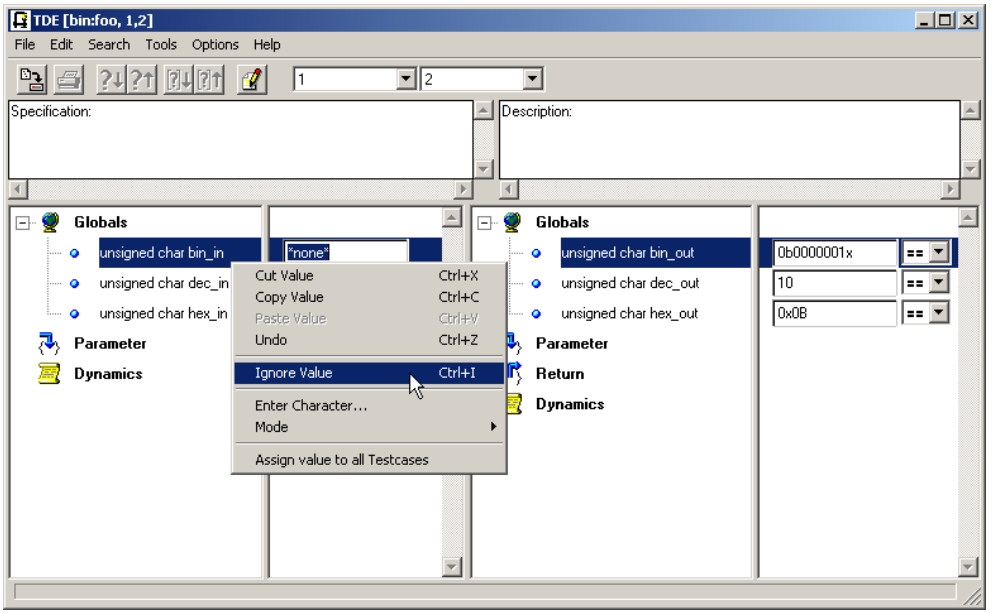
`0b000xxxx1111000`

- **`0b`** is the format specifier (s. Enter Values in Decimal, Hexadecimal or Binary Format on page 234)
- **`0`** denotes an unset bit
- **`1`** denotes a set bit
- **`X`** denotes a don't care bit in the expected result.



Ignore Values for a Test Step

By default, all variables have to be assigned a value if the passing direction of an element has been set to `In` or `InOut` in the **TIE.** However, it can be useful to not overwrite a value calculated in the last test step. In this case you may use the special value **`*none*`**.



The value defined for the last test step will remain as it is. If the value ***none*** is used in the first test step, global variables and parameters remain none-initialized. In this case the actual content of these variables will depend on the initialization of the compiler used.

View Results after Test Execution

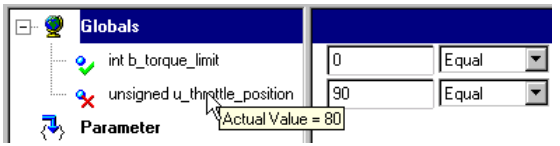
Tessy will compare both expected and actual values after test execution. The result will be either failed or passed.

The TDE will display results as follows:

A green tick will indicate that the actual value coincide with the expected value for the given variable with respect to the evaluation mode.

A red cross will indicate that the actual value do not coincide with the expected value for the given variable with respect to the evaluation mode.

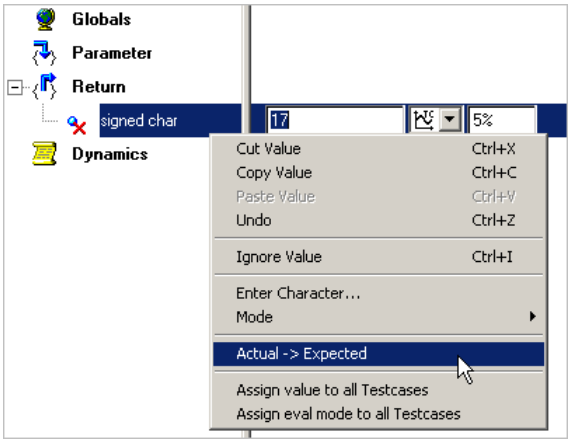
- If you move the mouse pointer over a variable, the actual value will be displayed as a tool tip.



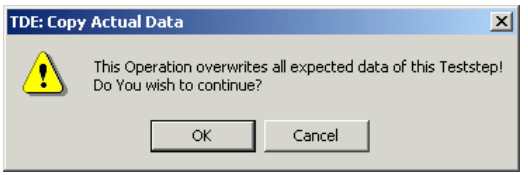
Adopt Test Results as Expected Values

TDE provides an option to take over actual value as expected value from the last test run. You may assign actual values for a single variable or the whole test step.

- Select a variable and Choose **Actual -> Expected** from the context menu. *TDE will use the actual value as expected value.*



- To take over actual values for the whole test step choose **Actual -> Expected** from the **Edit** menu. *A dialog will indicate that your expected data will be overwritten.*



If you choose **OK**, TDE will take over the actual values as expected values into the expected values fields.

Copy and Paste Test Data

Cut, **Copy** and **Paste** are functions that allow you to move or copy test data from one test step, variable, structure and array (etc.) to another one within the **TDE**.

- You can use these functions either through the context menu of the interface element, through the **Edit** menu, or by using shortcuts:

  `[Ctrl]+[x]` > will cut test data.
  `[Ctrl]+[c]` > will copy test data.
  `[Ctrl]+[v]` > will paste test data.

***Note**: Copying and pasting test data is only possible with type compatible interface elements, arrays of the same type and size and so on.*

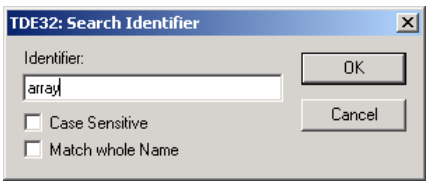
- If you want to insert values directly from the clipboard or if you want to copy values into the clipboard you must use the following shortcuts:

  `[Shift]+[Insert]` > will insert test data from the clipboard.
  `[Shift]+[Delete]` > will copy test data into the clipboard.

Search for Variables

If the interface of your test object is more complicated you may use the **Search** dialog of **TDE**.

- Select **Search Identifier** from the **Edit** menu or use `[Ctrl]+[F]`. *The Search Identifier dialog will open.*



- Enter your search term and choose following options if necessary:

- Case Sensitive
- Match whole name

- Click **OK**.

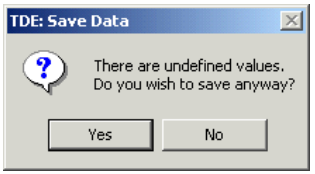
  If you want to search for the same term again, press [F3].

Exit the TDE

Before you can close the TDE you have to save your settings.

- Either click on **Save** in the toolbar or choose **Save** from the **File** menu.

  If you have still undefined values, following error message will appear.



- Select **Exit** from the **File** menu, or choose [ALT]+[F4].

Running a Test

Steps to Perform

After configuring the test environment and the test data for a particular test object, you're now ready to execute the test. During this process, Tessy will perform the following steps:

- Generate the test driver based on the interface information and user code provided.
- Link the test driver to the test object to create an executable file.
- Run the test.

Test data is only transferred to/from the target whenever a test is run. Depending on the size of the interface and the number of test cases to run, the entire test run process could take a few minutes to complete.

Refer to the respective application notes for details on how to configure a specific target system for the test execution, e.g. HiTOP, TRACE32. Some of the target debuggers need to be started prior to test execution whereas others will be started from Tessy automatically.

***Note**: All application notes are available within the documentation folder of your Tessy installation or just choose **Documents** from the **Help** menu.*

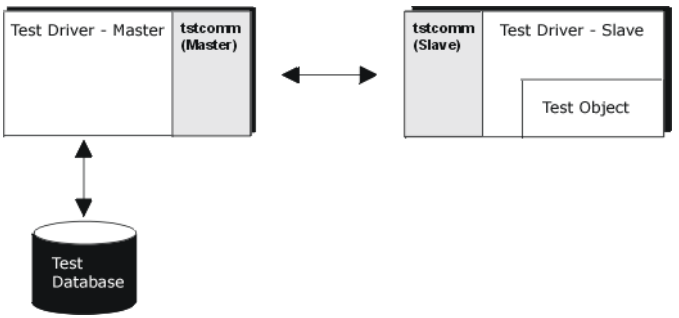
Structure of the Test Driver

The generated test drivers are based on a master/slave concept:

The **Test Database** will be accessed by the **Test Driver Master**. During a test run, the Master sends test data to the **Test Driver Slave** via its `tstcomm` interface. The implementation of this interface depends on the kind of target system used.

The **Slave** contains the **Test Object** and the test driver code to fill the input data and read out the actual results. This code is compiled using the respective target compiler and the resulting binary will be executed on a target system. This may either be a simulator, debugger or emulator. In case of the gcc compiler (delivered with Tessy), the slave will run as a normal windows process.

The **Slave** executes the test object with the input data from the test database and returns the results to the **Master**. The **Master** then stores these results into the test database.



At the end of the test run, the test evaluation takes place:

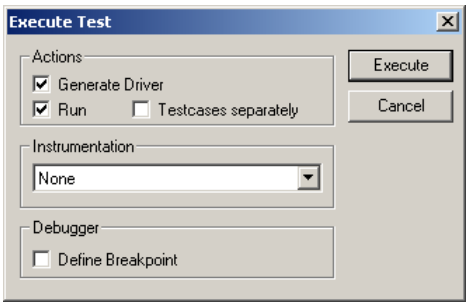
- The actual results will be compared with the expected values according to the evaluation mode.
- The last step of this cycle is the generation of an XML result file. This file will be used for reporting.

Setting Test Options

***Note**: You have to select at least one test case before running a test. Otherwise, the **Run** check box will be disabled (see below).*

To start the test run, do the following:

- Either click on **Execute Test** in the toolbar or select **Execute Test** from the context menu of the respective test object. *The Execute Test dialog will appear to allow various options to be set.*



Generating the Test Driver

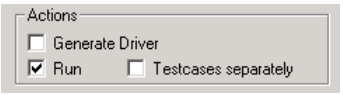
Tessy automatically indicates which steps are necessary in order to be able to run a test by ticking the appropriate check boxes within the **Action** panel.

During the test run process, Tessy will perform the steps as follows:

- Generate the test driver based on the interface information and user code, compile and link with the test object to produce an executable file.
- Run the test

***Note**: Check **Generate Driver** to force a rebuild of the test driver in case of changes of the test environment that Tessy may not yet have been detected (i.e. changes to header files).*

Once the test driver has been built and compiled, it can be run as often as required. You may select a subset of your test cases and run the test again by just selecting the **Run** option. Changes to test data and expected results may require building a new test driver, resulting in **Generate Driver** to be checked automatically by Tessy.

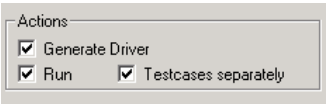


Running Test Cases Separately

Use the option **Testcases separately** to specify that a separate test run is to be performed for each test case. The normal behavior is that the test binary is downloaded to the target and all selected test cases (and test steps) are executed one after another. Internal states of the test object may then influence the results of successive test cases (e.g. you may get different results for test case 2 when either running test cases 1 and 2 or running test case 2 only).

The option **Testcases separately** avoids such problems associated with variable initialization, since the test driver slave will be restarted for each test case (or reloaded to the target environment). This results in an initial state of memory and variables before execution of each individual test case.

When running a test on a target debugger or emulator with this option set, the binary file is downloaded to the target device for each individual test case. Because of these additional downloads, the overall test execution will be slowed down.



***Note**: The **Testcases separately** option is stored individually for each test object. The initial setting of this option within the Execute dialog depends on the setting stored for the selected test object. For batch test execution (s. Batch Test), this option will also be used as stored for each individual test object.*

Coverage Analysis

Tessy supports both **C1** and **C2** coverage measurements. Since there are different meanings associated with these coverage measurement names, we will shortly explain our definition of these measurements:

We distinguish between **C1** (branch / decision) coverage measurement, **MCC** (Multiple Condition Coverage) and **MC/DC** (Modified Condition / Decision Coverage). The least two measurements are referred to as **C2** coverage measurement. All three measurements may be used in conjunction when running tests.

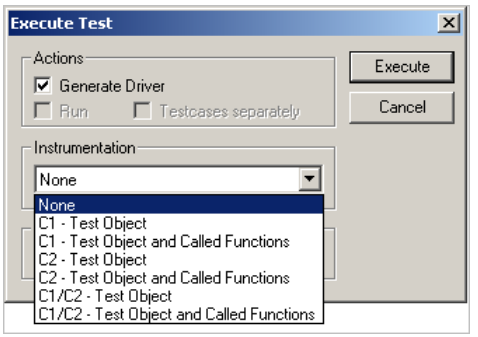
The **MC/DC** coverage includes the calculation of the required atomic value combinations for each condition within the test object. This helps developers in finding the required combinations for full **MC/DC** coverage. **MC/DC** in Tessy supports branch points such as in if or while statements.

For more information on this topic, please refer to

- Liggesmeyer, P., *Testen, Analysieren und Verifizieren von Software*, Software Qualität, Spektrum, Akademischer Verlag, ISBN: 978-3-8274-1118
- Cornett, S., *Code Coverage Analysis*, Bullseye Testing Technology, 1996-2005, www.bullseye.com/coverage.html

---

The **Instrumentation** combo box of the **Execute Test** dialog provides following options:



- **None**
  No instrumentation will be performed.
- **C1 Test Object**
  Branch coverage testing will be performed (only the test object code will be instrumented).

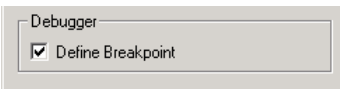
  C1 Test Object and called Functions
  Branch coverage testing will be performed for the test object itself and for all functions called by the test object.
- **C2 Test Object**
  Multiple Condition Coverage **MCC** and Modified Condition / Decision Coverage **MC/DC** testing will be performed for the test object (only the test object code will be instrumented)**.**

  C2 Test Object and called Functions
  Multiple Condition Coverage **MCC** and Modified Condition / Decision Coverage **MC/DC** testing will be performed for the test object itself and for all functions called by the test object.
- **C1 / C2 Test Object**
  C1 / C2 Test Object and called Functions
  Both test measurements will be applied as described above.

The results of every coverage measurement can be reviewed in the coverage viewer as soon as the test was carried out. For details refer to the chapter “View Coverage Results“.

Setting Breakpoints

If a test case returns an unexpected result, debugging of the test object is possible by using the **Define Breakpoint** option.



If this option has been enabled, Tessy will define a breakpoint at the beginning of the test object during test run. This will effect that the test run will be stopped after down loading the test binary into the debugger and running until the beginning of the test object code.

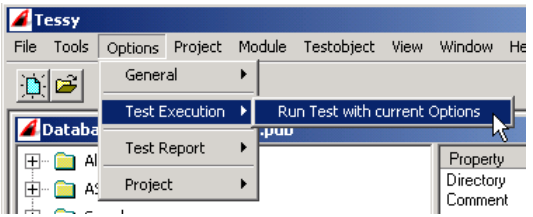
The **Define Breakpoint** option allows you to jump start into debugging and step through the test object for troubleshooting.

Note: *This option may not be available on certain target debuggers or emulators that do not support interactive debugging when running automated tests.*

Run Test with Current Options

You can start your testing immediately using the last settings of the **Execute Test** dialog by default.

- Choose **Run Test with current Options** from the **Options|Test Execution** menu.



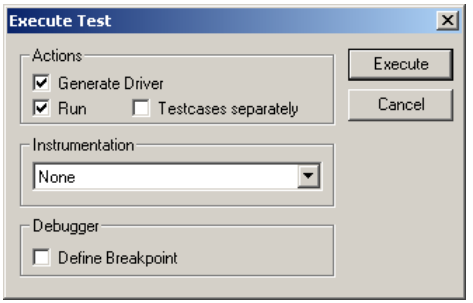
This will prevent the **Execute Test** dialog from appearing and allowing test execution to start immediately.

Execute Test Run

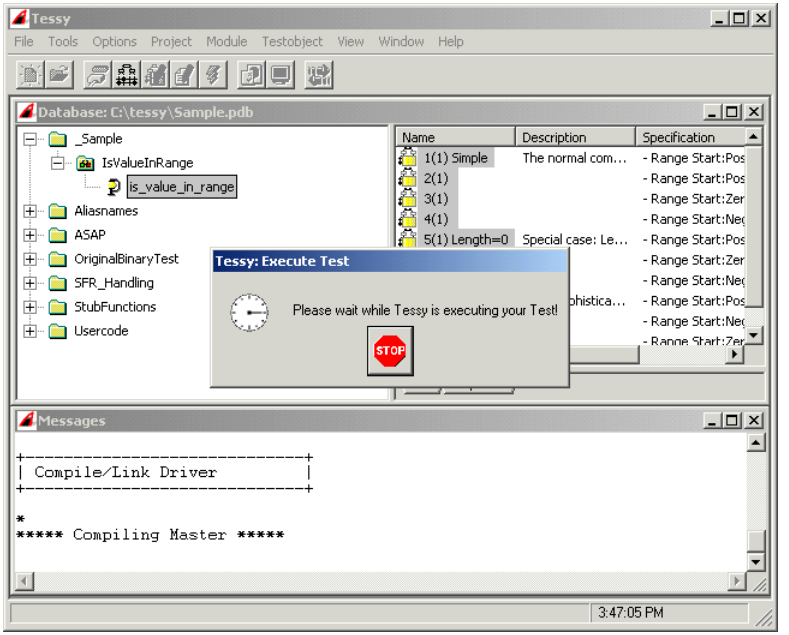
Potentially you have to launch your debugger first before you can start the test execution in Tessy. Please refer to the appropriate application notes for your target environment under **Help|Documents|Targets**.

Once all settings have been completed (see Setting Test Options) you can start the test as follows:

- Choose **Execute** from the **Execute Test** dialog.



The message "Please wait while …" will appear.



Note: *While executing the test, you will see the progress and any errors within the message window of Tessy.*

Tessy will do the following, while executing the test:

- Create and compile the test driver and link it with the user code and the test object to the test executable (Option: **Generate Driver**)

- Start the test driver master and load the test driver slave to the target environment to execute the test object (Option: **Run**).

***Note**: The test execution may be canceled at any time by clicking the **STOP** button.*

At the end of the test run, both the test object icon and the test case icon(s) will change their colors either to **Green** (passed) or **Red** (failed) (see Symbol Appearances).

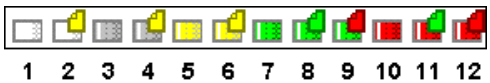
You may open the **TDE** or **UCE** to view the test results (see View Results after Test Execution respectively Usercode). You may also create a test report for this purpose (see Creating Reports).

Symbol Appearances

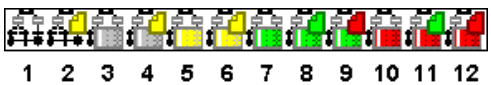
Tessy displays different test states with colored symbols. Both, test case and test object icons will change their appearance during the process of test preparation and whenever a test run is completed.

Tessy distinguishes 12 test case states and two kinds of test case symbols: Test cases that are created by Tessy and test cases that are imported from CTE (see pictures below).

Test cases created by Tessy



Test cases imported from CTE



Furthermore, the test object icon will also change its appearance, depending on the test case state or test result, e.g., the green test object icon is used as an indication that all tests have passed successfully. The following four states are possible (see picture below).

Test Object states

***Important note**: If the test object interface has still unresolved interface objects, Tessy will not open the test object clamp to indicate that. You will not be able to insert test cases. In that case, please open the test interface editor TIE to update the status.*

The table below lists all possible test case states and their meanings:

Test case status	Description	Test object status
1 -	Test case inserted but not initialized (TDE).	1 -
2 -	Test case inserted but not initialized (TDE). Eval Macros inserted and/or Function Call Trace used (UCE).	1 -
3 -	Test case inserted and partially initialized (TDE).	1 -
4 -	Test case inserted and partially initialized (TDE). Eval Macros inserted and/or Function Call Trace used (UCE).	1 -
5 -	Test case inserted and initialized (TDE).	2 -
6 -	Test case inserted and initialized (TDE). Eval Macros inserted and/or Function Call Trace used (UCE).	2 -
7 -	Test is successfully executed (TDE). Expected values specified in TDE are matching with the actual result values.	3 -
8 -	Test is successfully executed. Expected values specified in TDE are matching with the actual result values. Eval Macro results are as expected and/or	3 -

	Function Call Trace are matching with expected values (UCE).	
9 -	Test is not successfully executed. Expected values specified in TDE are matching with the actual result values, but Eval Macros and/or Function Call Trace are not matching with expected values (UCE).	4 -
10 -	Test is not successfully executed (TDE). Expected values specified in TDE are not matching with the actual result values.	4 -
11 -	Test is not successfully executed. Expected values specified in TDE does not coincide with the actual values, but results with Eval Macros and/or Function Call Trace does agree with expected values (UCE).	4 -
12 -	Test is not successfully executed. Expected values specified in TDE are not matching with the actual result values. Also Eval Macros and/or Function Call Trace results are not matching with expected values (UCE).	4 -

UCE = Usercode Editor, TDE = Test Data Editor

***Note**: A test case can not be executed before initialization (- yellow state). If a test case could not be executed (i.e. because the test was aborted), it will remain yellow.*

Change of the Test Case Status

An executed test case with status green or red will change the status to yellow after the following actions:

- user code has been inserted or changed (all test cases will get the yellow status)
- test data have been changed (only the affected test cases will get the yellow status)
- a new test case has been inserted (all other test cases will get the yellow status)
- a new test run has been executed, but the test case has not been selected for that test run.

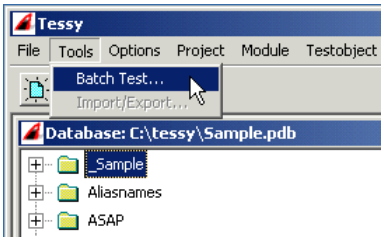
Your test cases will change to status gray after an assignment of test data by using IDA or if you change the settings of the interface (changing passing directions of variables). You may then select all test cases and initialize them by choosing **Initialize Testcase** from the test case context menu if no further test data has to be inserted.

Batch Test

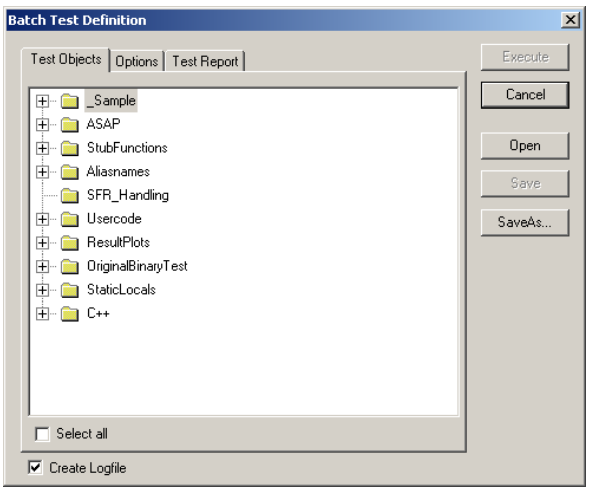
Tessy provides a **Batch Test** feature to execute all tests at once. This allows you to select whole projects, modules or test objects for testing.

To create a Batch Test, do the following:

- Open a test database and choose **Batch Test** from the **Tools** menu.



The Batch Test Definition dialog will open.

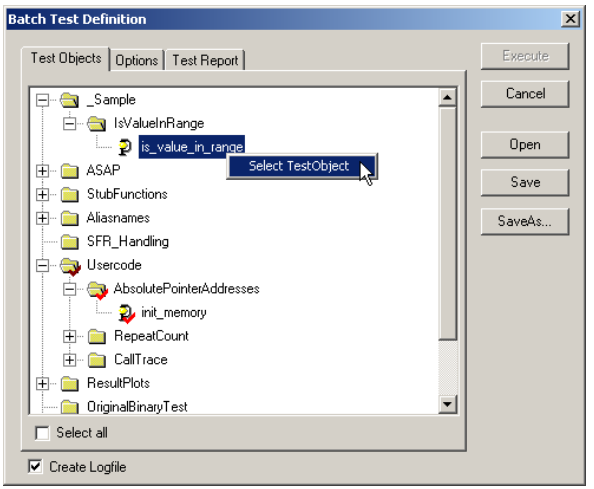


Test Objects Tab

The Test Objects Tab displays the complete project tree as shown in Tessy. You may browse through the tree and choose an item as follows:

- Select a project, a module or a test object and choose **Select Entire Project** from the context menu (respectively **Select Entire Module** or **Select Test Object**). Alternatively, press `[Space]` to select an item.

  Tessy will mark all selected items by a red tick.



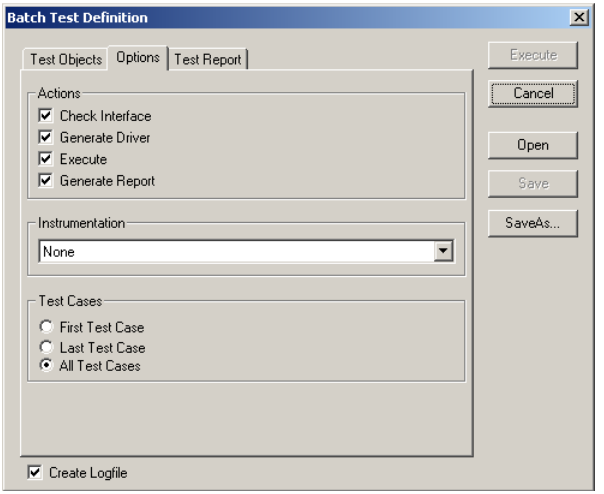
Options Tab

The Options Tab is similar to those offered for an individual test run (see Setting Test Options for details).

Following additional options are available:

- Check Interface
  The interface checking will be omitted if the check-box is disabled.
- Generate a Report
  The report will be generated for the entire batch test run.

- Test Cases
  Only the first, the last or all test cases will be executed.



***Tip**: You may choose only "Generate Report" to create a batch report without running the test again. The current results of all previously executed tests will be used for report creation. This is useful, if you ran a batch test and some test objects or modules failed to execute: You may then just run the not executed tests again (start them manually) and create a report for the whole batch test afterwards.*

***Note:** Unlike previous versions of Tessy (v2.3 and earlier) the "Execute Testcases Separately" option has been removed from this dialog. The batch test will now use this setting of the Execute Test dialog for every test object individually (because this settings is now stored for each test object individually).*

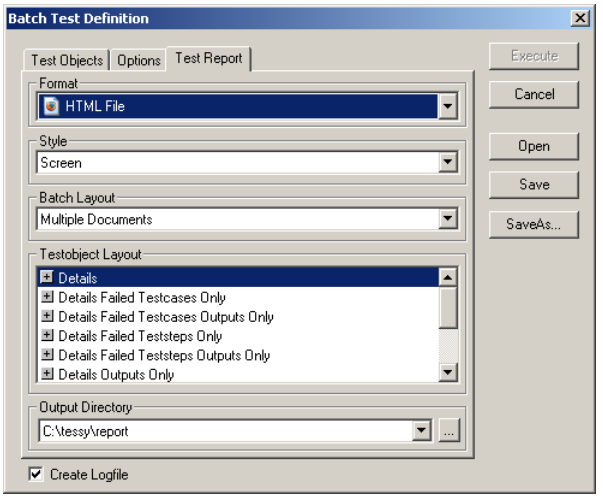
Test Report Tab

The Test Report Tab is nearly similar to those offered for an individual test run (see Test Report for details)

Following additional options are available:

- Batch Layout

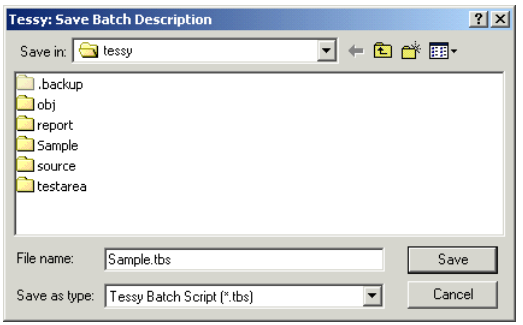
- Test Object Layout



Note: *The batch test report will be stored in a directory tree corresponding to the projects, modules and test objects executed during the batch test run. The whole report structure will be created in a subdirectory that will be named using the batch test name.*

Save and Execute Batch Test

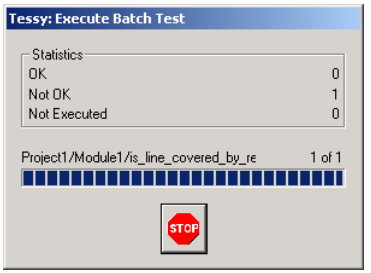
When you have completed all settings, click on **Save** to save your settings to a file. *The Save Batch Description dialog will open.*



- Enter a name and click **Save**. By default, batch test files have a **tbs** extension.

Now you can run the test.

- Click **Execute** to start the test execution.
  The following dialog will appear.



- Alternatively, you may use the command line prompt to start the test execution (see Running Batch Tests at Command Line).

***Note**: The test execution may be canceled at any time by clicking the **STOP** button. While executing the test, you will see the progress and any errors within the message window of Tessy.*

View Batch Test Report

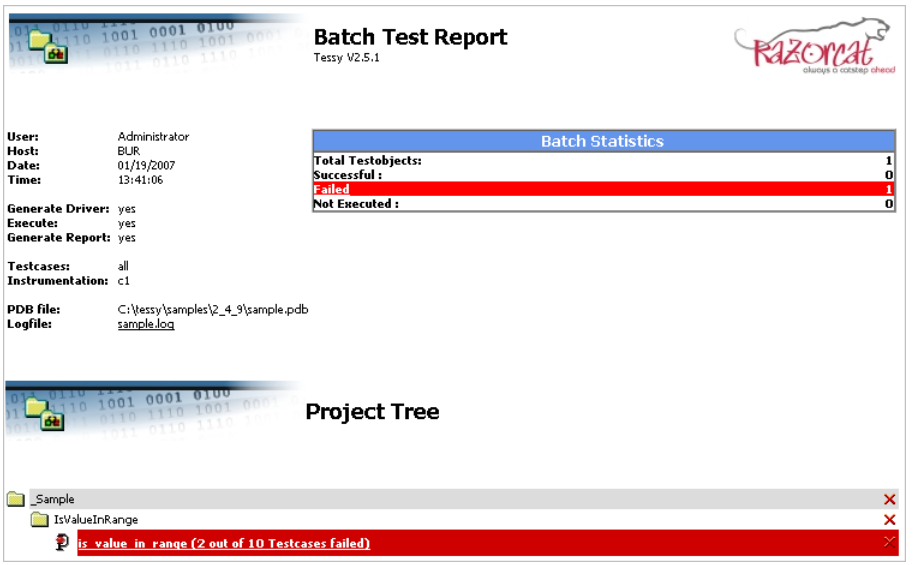
After finishing the batch test, Tessy will start an appropriate viewing tool depending on your local windows settings. In case of HTML reports, your default web browser will be started.

Structure of HTML Batch Reports

The HTML batch report will be created with the following structure:

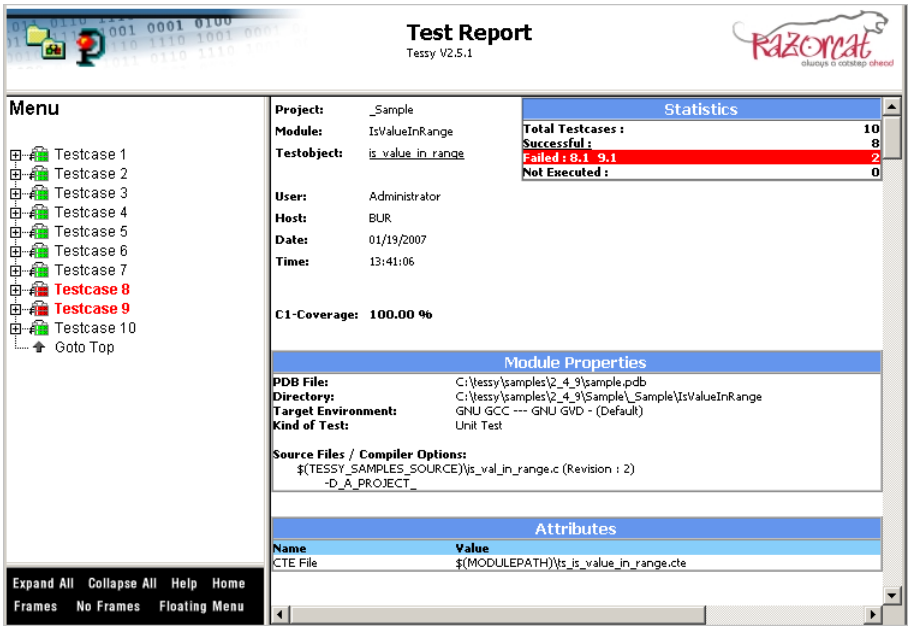
- **Batch Statistics**
  Displays a consolidated statistics for all test objects.
- **Batch settings section**
  Displays all settings for test execution, e.g. `Generate Driver`.
- **Project Tree**
  Displays all projects, modules and test objects. The green/red icons mark the passed/failed status of the test objects.

  Each test object will be linked to the corresponding test object report, e.g. `is_value_in_range`.



You may click the link to open the corresponding test object report within the browser (refer to Test Report for details).

***Note**: Use the **Dropdown** list from the **Back to** button of your browser to return to the **Batch Test** report.*



***Note**: The HTML report use relative links to all linked items. Therefore, you may zip the whole directory structure (with all files and folders) to provide a test report to other users for reviewing without problems. You may also use Compiled HTML Help File format to reduce the files to be copied (refer to Test Report for details).*

Running Batch Tests at Command Line

Tessy provides an interface to run a given batch test without the GUI of Tessy. This will assume that you have finished the test preparation creating a batch test file (see Batch Test).

The batch test file contains the name of the PDB file (with full path name) that shall be used, the execution settings and a list of projects/modules/testobjects to be executed. In order to overwrite (or ignore) the PDB file name within the batch test file, you may specify the PDB file on the command line.

Usage:

```
tessycmd [--log log-file] <tbs-file> [pdb-file]
```

- `--log` (optional)
  Note: Tessy creates also a log file `tbs-file-name.txt` by default in the directory where the `tbs-file` lies.
- `tbs-file`
  `*.tbs`, created using **Tools|Batch Test...**
- `pdb-file` (optional)
  This setting overwrites the name of the PDB file stored within the given `tbs-file`. It is useful, if you checked out your project into another location on disc (e.g. on another PC) and want to execute a previously created `tbs-file.`

Example:

This example assumes that `sample.tbs` already exists.

- Open a command prompt window and type the following:

  ```
  tessycmd --log c:\tessy\sample.log c:\tessy\sample.tbs
  ```

  The messages which normally appears in the message window of Tessy will also appear in the command prompt window. You may redirect the messages into a file using the standard windows command shell mechanisms.

Note: *tessycmd must be in your PATH or you must change into the bin folder of your Tessy installation, e.g.* `.\tessy_2.6\bin.`

`sample.log`

```
Batch Execution started at 22.07.2008 11:01
Project: _Sample
```

```
  Module: IsValueInRange
    Testobject : is_value_in_range
    Generating driver files
    Compile and Link Testdriver
    Executing Test with all Testcases.
    Creating XML result file
Batch Execution finished at 22.07.2008 11:02
```

Note: *Potentially you have to start your debugger/emulator/simulator at first to run the batch test, e.g. TRACE32.*

Test Report

Creating Reports

After every test run, Tessy creates a XML result file within the respective module folder. The report interface of Tessy generates a test report by using this XML file.

You may generate a test report as follows:

- After a test run was finished (Generating a Report after a Test Run).
  Tessy will automatically use the XML file from the respective module folder.
- By specifying a XML result file (Generating Reports using XML Result Files).
  In this case you don't need to open a project database to create a test report.

There are some predefined formats for the test report available:

- **`HTML, MS Word, MS Excel, RFT, CHM`** and **`ASCII`**.

***Note**: You may create your own Formats by using Python scripts. Please refer to the application notes Report Scripts from the **Help/Documents/Customization** menu for more information on this topic.*

Furthermore, it's possible to view the results of a C1 and/or C2 (MCC, MC/DC) coverage test using the monitoring function.

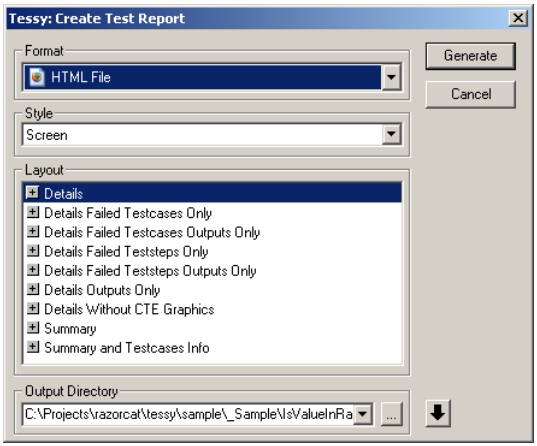
- see chapter Monitoring of C1 and C2 for more details.

Generating a Report after a Test Run

A test report can be generated as soon as Tessy has finished running a test. Tessy will automatically use the generated XML file from the respective module folder to create the report.

- Click **Test Report** icon in the toolbar or select **Test Report** from the context menu of a test object to create a test report.

  The Create Test Report dialog will open.



- You may choose from the following settings:

  Format
  Format specifies the file format of the test report. You may choose between HTML, ASCII text, Microsoft Word, Microsoft Excel or other available formats.

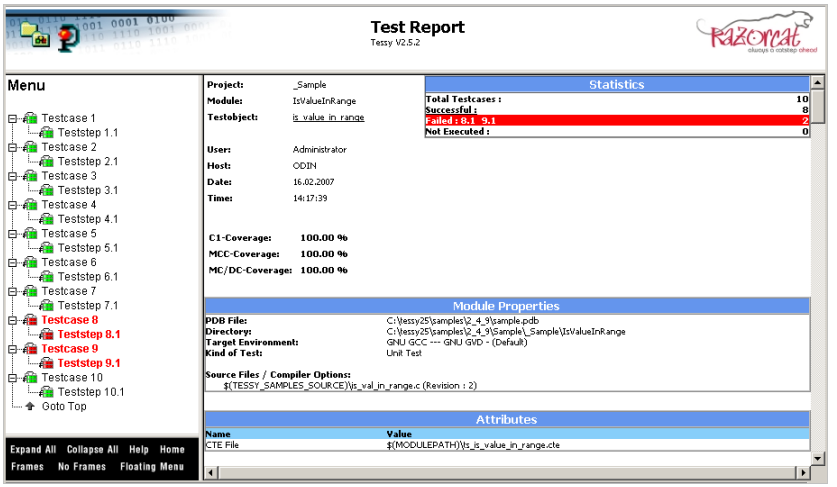
  Style
  Sets the style of the document. There are some styles available depending on the file format used.

  Layout
  Sets the layout of the test report. Various layouts are available depending on the file format used.

  Output Directory
  In this box, you may choose another directory for the test report. By default, the report will be created into the module directory.

- After selecting of suitable settings (e.g. for HTML documents), click **Generate** to generate and view the test report.

  Tessy will start an appropriate viewing tool depending on the file format used. In case of HTML reports, your default web browser will be started.

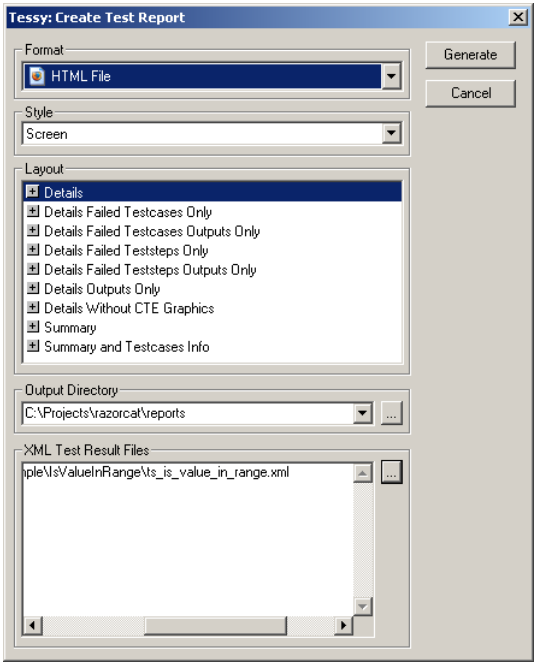


Generating Reports using XML Result Files

You may use already generated XML files to create test reports. In this case you don't need to open any project databases to create the reports.

- Open Tessy and click the **Test Report** icon in the toolbar.

  The Create Test Report dialog will open.



- You may choose from the following settings:

  Format
  Format specifies the file format of the test report. You may choose between HTML, CHM, ASCII text, Microsoft Word, Microsoft Excel or other available formats.

  Style
  Sets the style of the document. There are some styles available depending on the file format used.

  Layout
  Sets the layout of the test report. Various layouts are available depending on the file format used.

  Output Directory
  Sets a root folder in which the reports will be generated.

- To specify a directory, click the path selection button.

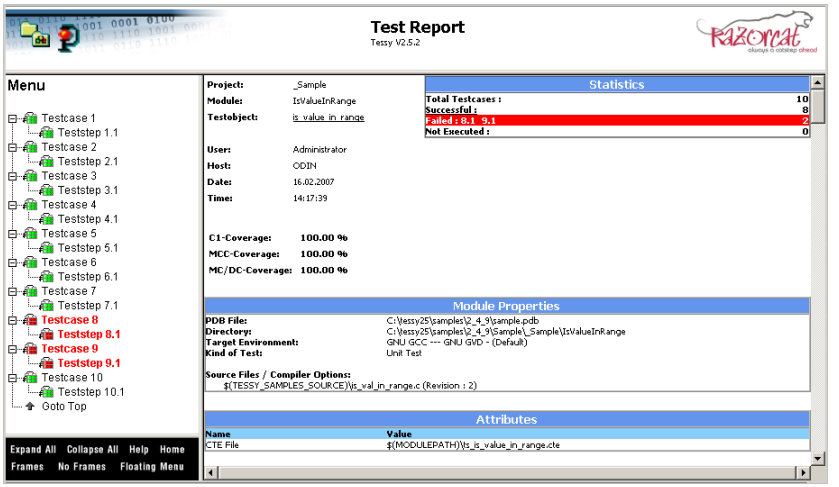
  XML Result Files
  At least one XML result file has to be selected to generate a report.

- To select a XML file, click the path selection button.

- After selecting of suitable options, click **Generate** to generate the test report.

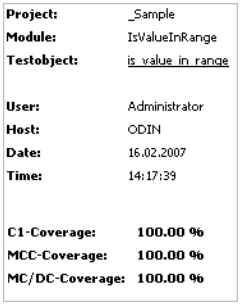
Test Report Structure

The appearance of a test report depends on the file format and layout used. Therefore, the following description will assume that HTML was used (Layout: *Details*).


Test Report
Tessy V2.5.2
Menu
Testcase 1
Teststep 1.1
Testcase 2
Teststep 2.1
Testcase 3
Teststep 3.1
Testcase 4
Teststep 4.1
Testcase 5
Teststep 5.1
Testcase 6
Teststep 6.1
Testcase 7
Teststep 7.1
Testcase 8
Teststep 8.1
Testcase 9
Teststep 9.1
Testcase 10
Teststep 10.1
Goto Top
Expand All Collapse All Help Home
Frames No Frames Floating Menu
Project: _Sample
Module: IsValueInRange
Testobject: is_value_in_range
User: Administrator
Host: ODIN
Date: 16.02.2007
Time: 14:17:39
C1-Coverage: 100.00 %
MCC-Coverage: 100.00 %
MC/DC-Coverage: 100.00 %
Statistics
Total Testcases : 10
Successful : 8
Failed : 8.1 9.1 2
Not Executed : 0
Module Properties
PDB File: C:\tessy25\samples\2_4_9\sample.pdb
Directory: C:\tessy25\samples\2_4_9\Sample_Sample\IsValueInRange
Target Environment: GNU GCC --- GNU GVD - (Default)
Kind of Test: Unit Test
Source Files / Compiler Options:
${TESSY_SAMPLES_SOURCE}\is_val_in_range.c (Revision : 2)
Attributes
Name
Value
CTE File
${MODULEPATH}\is_is_value_in_range.cte

General Data

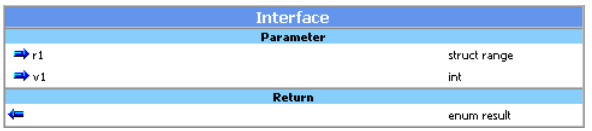
The **General Data** section (right hand window upper left corner) contains following information (see picture below):



- **Project**
  Name of the project.
- **Module**
  Name of the module
- **Testobject**
  Name of the test object of the test run. You will get the interface description if you click the link - see Interface Description, below)
- **User**
  Name of the user who carried out the test run.
- **Host**
  Name of the computer on which the test was executed.
- **Date**
  Date of the test execution.
- **Time**
  Time of the test execution.
- **C1 :: MCC :: MC/DC Coverage**
  If you executed the test in the respective modes, the % coverage will be printed here.

Interface Description

You can view the interface description of the test object if you click the test object name in the **General Data** section. *Following page will open:*



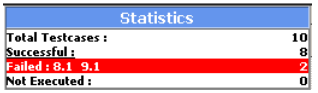
Interface	
Parameter	
r1	struct range
v1	int
Return	
	enum result

The interface table lists all kinds of elements used by the test object, e.g. Global variables, Parameter and Return Value (if any).

Next to each element you will find the passing direction for **Input** values, for **Output** values and for **InOut** values.

Test Case Statistics

In the upper right corner of the test report, the statistics of all test cases are listed. You will see the number of successful and failed test cases respectively test steps.



Statistics	
Total Testcases :	**10**
Successful :	**8**
Failed : 8.1 9.1	**2**
Not Executed :	**0**

- To view a failed test step, click the test step number, e.g. **8.1.** The browser will directly jump to this test step.

Module Properties and Attributes

Module Properties and **Attributes** are listed in their respective tables of the test report (see also Module Properties).

Module Properties	
PDB File:	C:\Projects\razorcat\tessy\Sample.pdb
Directory:	C:\Projects\razorcat\tessy\sample_Sample\IsValueInRange
Target Environment:	GNU GCC --- GNU GVD - (Default)
Kind of Test:	Unit Test
Linkeroptions:	0
Sources	$(PROJECTROOT)\scr\is_val_in_range.c (Revision 2)
[Compileroptions]:	

Attributes	
Name	**Value**
CTE File	$(MODULEPATH)\is_value_in_range.cte

Displaying Keywords $Revision$, $Author$ and $Date$

Version Control Systems provide keyword expansion to embed version control information into text files. Tessy will display such expanded keywords within the test report for the used source files.

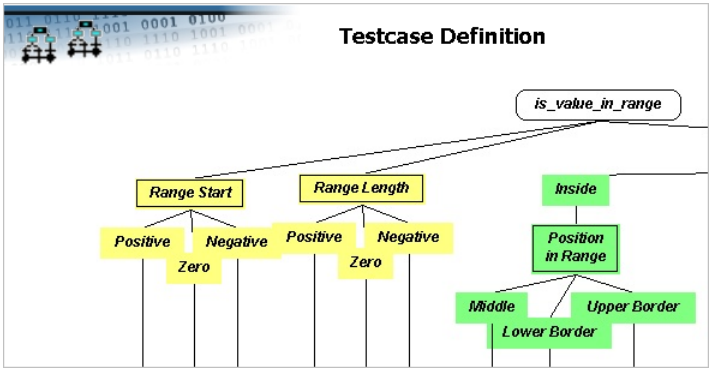
Following keywords are supported:

- $Revision$ (Revision number)
  Example: `$Revision 2$`
- $Author$ (User who checked in the revision)
  Example: `$Author: Konrad Lorenz$`
- $Date$ (Date and time stamp for the revision)
  Example: `$Date: 2/24/06 5:15:59$`

Module Properties	
PDB File:	C:\Projects\razorcat\tessy\Sample.pdb
Directory:	C:\Projects\razorcat\tessy\sample_Sample\IsValueInRange
Target Environment:	GNU GCC --- GNU GVD - (Default)
Kind of Test:	Unit Test
Sources	$(PROJECTROOT)\scr\is_val_in_range.c (Revision 2, Author Konrad Lorenz, Date 2/24/06 5:15:59)
[Compileroptions]:	

Testcase Definitions

The test report will also contain the structure of the classification tree in case you have used CTE for your test case definitions. Tessy will then insert a graphic for each sub-tree (refinements) of the classification tree.

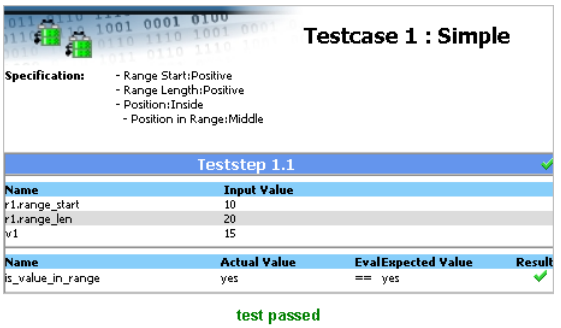


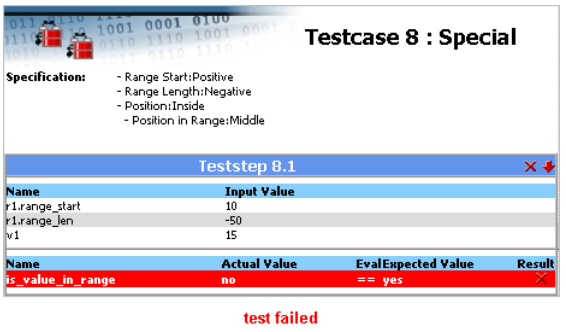
If you don't want to insert graphics, please choose "Details Without CTE Graphics" from the Create Test Report dialog (s. Generating a Report after a Test Run).

Test Cases

The **Details** layout of a report lists all test cases and test steps in their respective tables regarding their test step numbers.

A green tick ✔ or a red cross ✘ will indicate whether a test step was successfully executed or failed (**Result**).

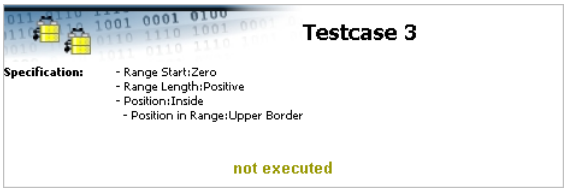




The previous ( ) or next ( ) arrows will direct you to the previous or next failed test step.

The rows of the test step table will be marked in red if an **Actual Value** didn't coincide with the **Expected Value** for the given **Evaluation** mode.

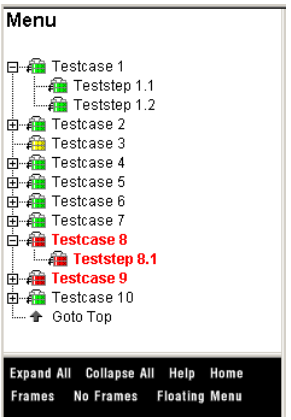
Not executed test cases will be displayed in yellow and have no tables attached.



Navigate Through the Test Cases

The navigation menu of the test report (left hand window) displays all executed and not executed test cases using a tree structure.

The green/red icons mark the passed/failed status of the test object. Yellow icons indicate that the test cases were not executed. By clicking on one item will display the corresponding test case/test step part of the report in the right hand window.



The caption of the menu means the following:

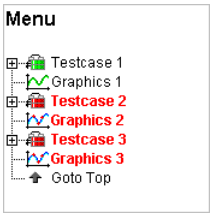
- **Expand all**
  Expands all test cases within the navigation tree.
- **Collapse All**
  Collapses all test cases within the navigation tree.
- **Home**
  Jumps at the top of the test report.
- **Frames**
  Shows the test report with the test case / test step navigation menu on the left side.
- **No Frames**
  Shows the test report without the test case / test step navigation menu.
- **Floating Menu**
  Undocks the navigation menu and shows it in a separate window. Simply click on the close window button of the navigation menu to switch it back into the main report window.
- **Help**
  Open a support request form (requires an internet connection). One of our Razorcat team members will contact you promptly.

Test Reports with Results Plots

The Result Plot integration allows to graphically visualizing the test results. This feature requires no further software to be installed (e.g. MatLab) but the special evaluation modes for result plots to be used (s. Enter Evaluation Mode).

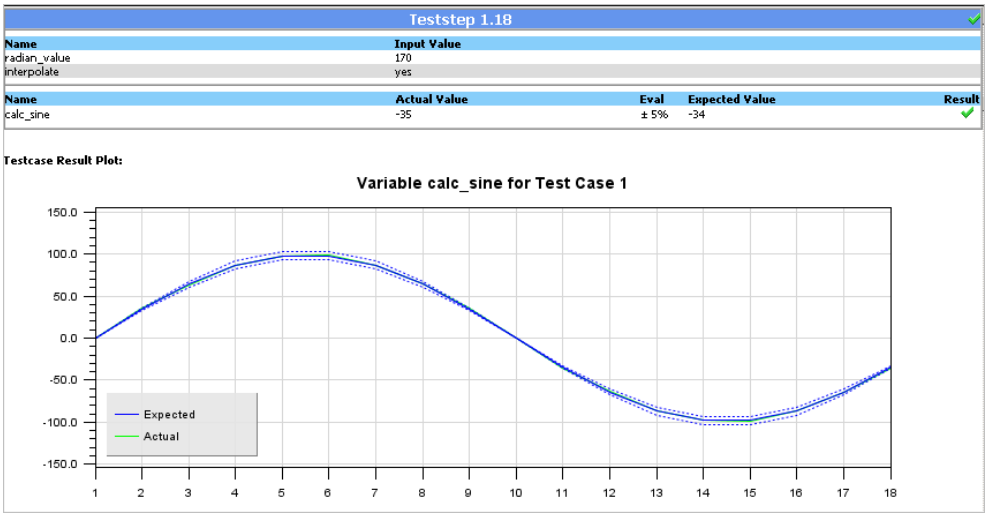
The preparation of a report with result plots takes place in the same manner as described before.

In addition, result plot reports contain graphics of the result plots and additional icons in the navigation menu.

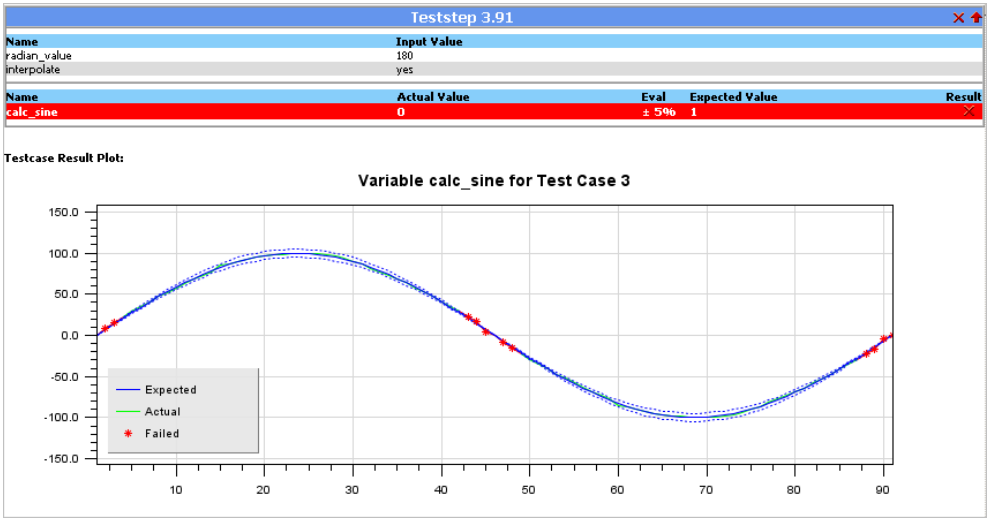


A green or a red graphics icon will indicate whether a test case was successfully executed or failed. Clicking on a graphics icon will display the corresponding result plot graphic in the right hand window of the test report.

Example: Testcase successfully executed



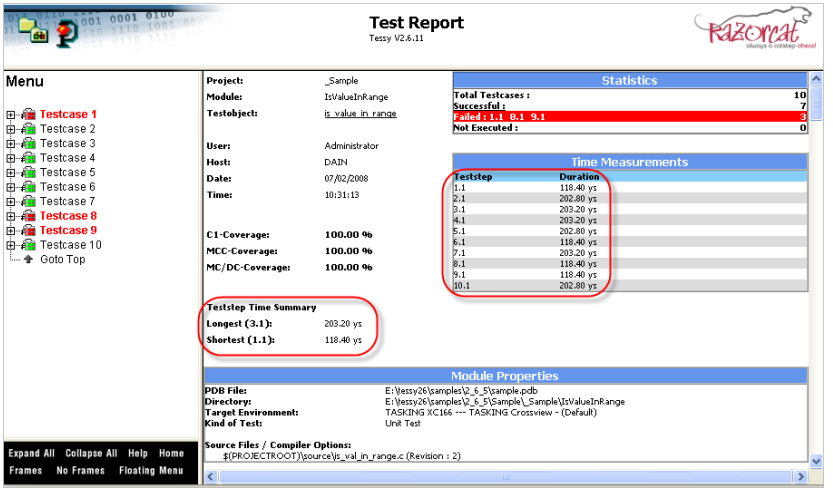
Example: Testcase with failed test steps



Test Reports with Timing Measurements

Tessy provides timing measurement implementations which are only available for selected targets and microcontrollers. Please refer to the application notes Timing Measurements under **Help|Documents** for more information on how to use timing measurements with Tessy.

The timing measurement results are displayed in the respective sections of the test report document, e.g. **HTML**


Test Report
Tessy V2.6.11
Menu
Testcase 1
Testcase 2
Testcase 3
Testcase 4
Testcase 5
Testcase 6
Testcase 7
Testcase 8
Testcase 9
Testcase 10
Goto Top
Expand All Collapse All Help Home
Frames No Frames Floating Menu
Project: _Sample
Module: IsValueInRange
Testobject: is_value_in_range
User: Administrator
Host: DAIN
Date: 07/02/2008
Time: 10:31:13
C1-Coverage: 100.00 %
MCC-Coverage: 100.00 %
MC/DC-Coverage: 100.00 %
Teststep Time Summary
Longest (3.1): 203.20 µs
Shortest (1.1): 118.40 µs
Statistics
Total Testcases : 10
Successful : 7
Failed : 1.1 8.1 9.1 3
Not Executed : 0
Time Measurements
Teststep Duration
1.1 118.40 µs
2.1 202.80 µs
3.1 203.20 µs
4.1 203.20 µs
5.1 202.80 µs
6.1 118.40 µs
7.1 203.20 µs
8.1 118.40 µs
9.1 118.40 µs
10.1 202.80 µs
Module Properties
PDB File: E:\tessy26\samples\2_6_5\sample.pdb
Directory: E:\tessy26\samples\2_6_5\Sample_Sample\IsValueInRange
Target Environment: TASKING XC166 --- TASKING Crossview - (Default)
Kind of Test: Unit Test
Source Files / Compiler Options:
${PROJECTROOT}\source\is_val_in_range.c (Revision : 2)

Monitoring of C1 and C2

View Coverage Results

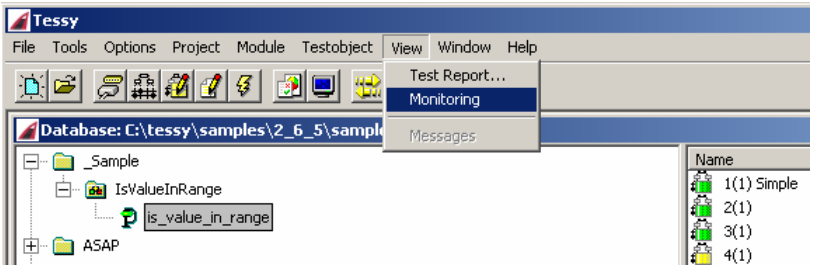
Tessy provides the following coverage measurements:

- C1 branch coverage
- MC/DC modified condition decision coverage
- MCC multiple condition coverage

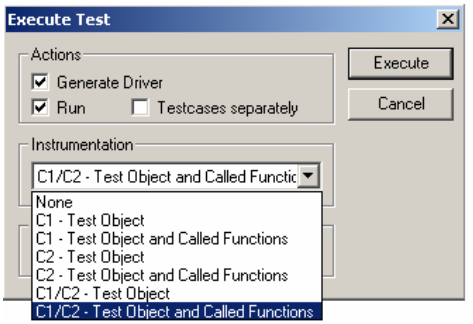
The least two measurements are summarized as C2 coverage within Tessy. You may select the desired coverage measurements before executing the test (within the Execute Test dialog). The coverage results are available for review in detail within the Tessy coverage viewer (**CV**) or summarized within the generated test reports.
The CV shows the results of the coverage measurement of a previously executed test. The available information displayed and the sub windows shown within the CV depend on the coverage options selected during the test run.
The CV will be updated with the coverage information of the currently selected test object whenever the **Monitoring** entry of the **View** menu is selected by the user.

Starting the Coverage Viewer CV

The coverage results are accessible from the **Monitoring** entry of the **View** menu. Within the toolbar, there is also a Monitoring button available.



The **Monitoring** menu entry is only enabled, if the previously executed test (for the currently selected test object) was carried out with one of the instrumentation options selected within the **Execute Test** dialog. The maximum information being displayed will be available with the **Instrumentation** option **C1/C2 – Test Object and Called Functions** like shown below:



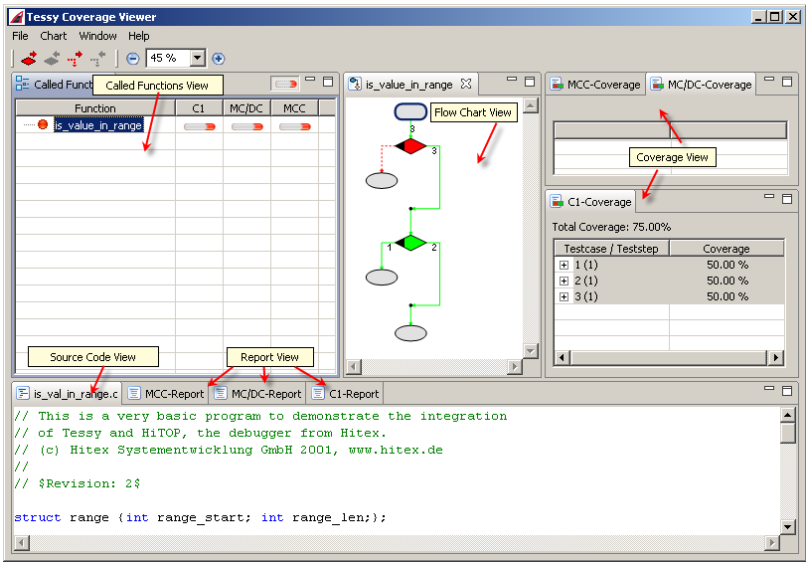
If you select **none** for the **Instrumentation** option within the **Execute Test** dialog, you will not be able to start the CV after the test run since there will be no coverage information available (for the currently selected test object).

Window Structure of the CV

The CV is a graphical tool based on the Eclipse framework. The different windows of the graphical user interface are called views. They may be moved, rearranged and closed freely. Please review the documentation of the Eclipse framework for further information on how to handle Eclipse applications in general.
In order to reset the outline of the CV to the default perspective, use the **Reset Perspective** item from the **Windows** menu.
The following views are available by default as shown below:

- The **Called Functions** tree view containing the test object itself and the functions called from the test object.

- The **flow chart** view showing the graphical representation of the control structure of the currently selected function.
- The different **coverage views** displaying the C1, MC/DC and MCC coverage results.
- The **source code view** showing the source code of the currently selected function (and highlighting selected decisions/branches).
- The **report views** showing the ASCII based summary reports for C1, MC/DC and MCC coverage.



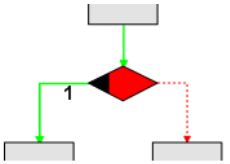
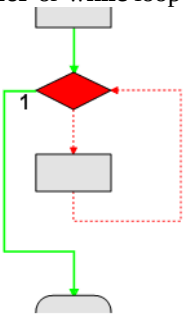
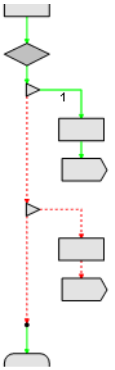
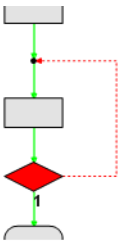
Flow Chart View

The flow chart view displays the code structure and the respective coverage in graphical form. For each function of the **Called Functions** view, the code structure will be displayed in a new flow chart view.

Within each flow chart, you will see the decisions and branches of the function being displayed. Green and red colors indicate whether the respective decision has been fully covered or the respective branch has been reached.

Displayed Elements

The following elements are displayed within the flow chart of the CV:

if decision 	The if branch on the left side was reached once, the else branch on the right side was not reached. The decision was not fully covered.
for or **while** loop 	The loop body was not reached, instead the exit out of the loop was executed once. The loop decision was not fully covered.
switch statement 	The first case branch was reached once, the second case branch and the default branch were not reached.
do while loop 	The loop body was only reached once (without repeated execution of the loop body branch) and the exit branch was reached. The loop decision was not fully covered.

Decision elements are either red or green, depending on the coverage result:

- If the decision was fully covered (MC/DC and MCC), it will appear in green
- If the decision wasn't fully covered, i.e. there is at least one condition combination that was not executed; the decision will appear in red.

If no C2 coverage has been selected for the last test execution, the decision elements remain grey, but they are still selectable in order to find the respective line of code in the source code view.

Selecting Elements

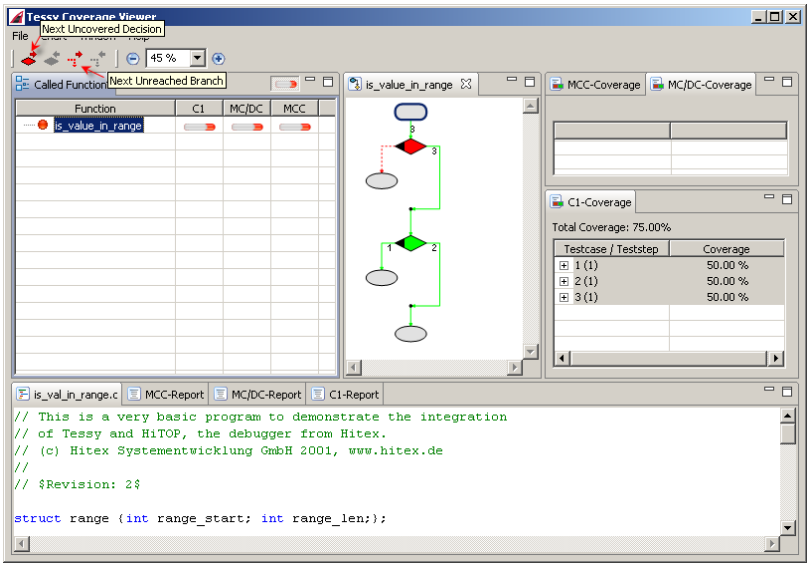
You may select decisions, branches and code statement elements within the flow chart. The respective code section will then be highlighted within the source code view.
Since not all connection lines within the flow chart are branches in terms of the C1 branch definition, some of the connection lines may not be selectable.

Zooming

The flow chart view may be zoomed using the toolbar buttons or the **Zoom In** or **Zoom Out** entries from the **Chart** menu.

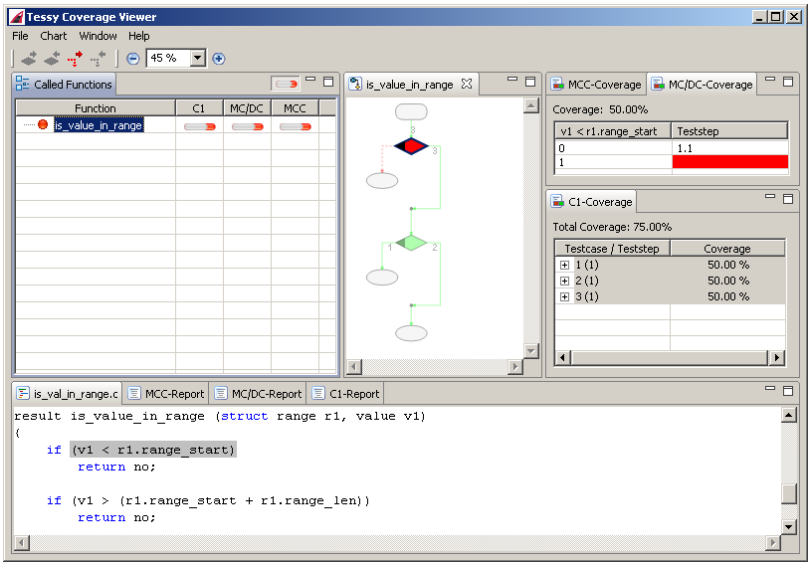
Searching for Uncovered Conditions or Unreached Branches

The CV provides search functionality for decisions and branches that are not fully covered respectively reached through all the executed test cases. The decisions and branches are already marked in red, but the search function may assist in finding all uncovered decisions or unreached branches.



Searching for Uncovered Conditions

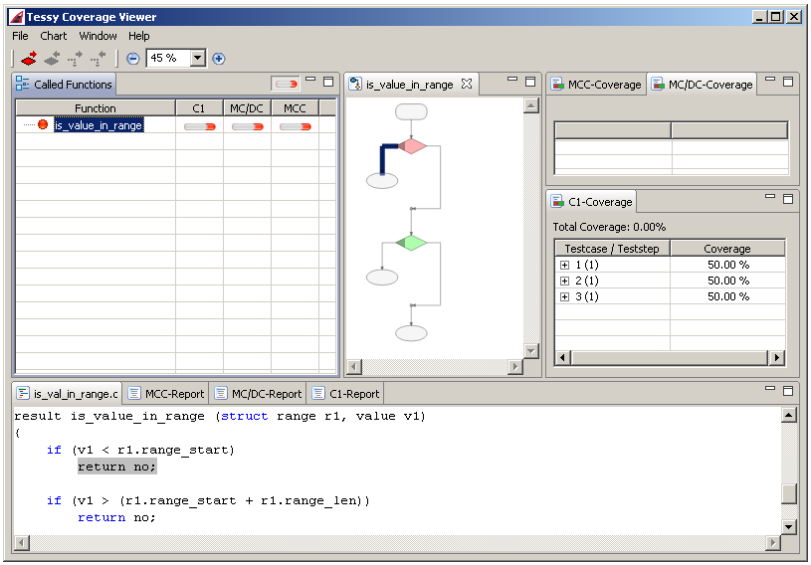
By clicking on the **Next Uncovered Decision** button within the toolbar or selecting the respective item from the **Chart** menu, the flow chart will change into the search result mode displaying the next uncovered decision found in normal mode while fading the rest of the flow chart as shown below:



Note: *To leave the search result mode of the flow chart tab, just click somewhere into the flow chart. CV will switch back into normal mode.*

Searching for Unreached Branches

By clicking on the **Next Unreached Branch** button within the toolbar or selecting the respective item from the **Chart** menu, the flow chart will change into the search result mode displaying the next unreached branch found in normal mode while fading the rest of the flow chart as shown below:

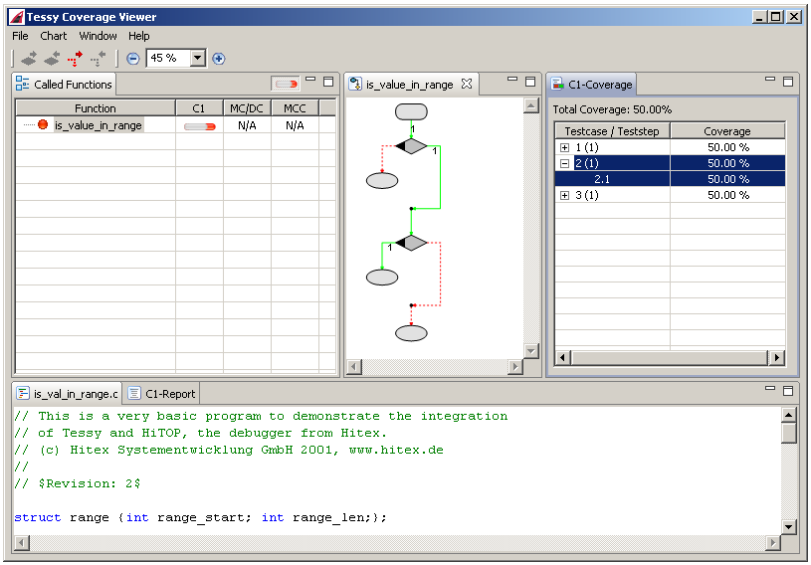


Note: *To leave the search result mode of the flow chart tab, just click somewhere into the flow chart. CV will switch back into normal mode.*

C1 Coverage View

The **C1-Coverage** view displays the branch coverage for each individual test case and test step as well as the total coverage for all test cases (and test steps). The coverage information in this view is always calculated for the currently selected function within the **Called Functions** view of the CV.

If you selected only the C1 coverage instrumentation for test execution, you will see only the C1 branches highlighted in red and green within the flow chart like shown below:



The total coverage

Summarized Total Coverage

The summarized total coverage is the relation between the total numbers of branches of the currently selected function compared to the number of reached branches. This coverage calculation includes the currently selected test cases and test steps within the **Testcase / Teststep** list. By default, all test cases are selected when opening the CV.

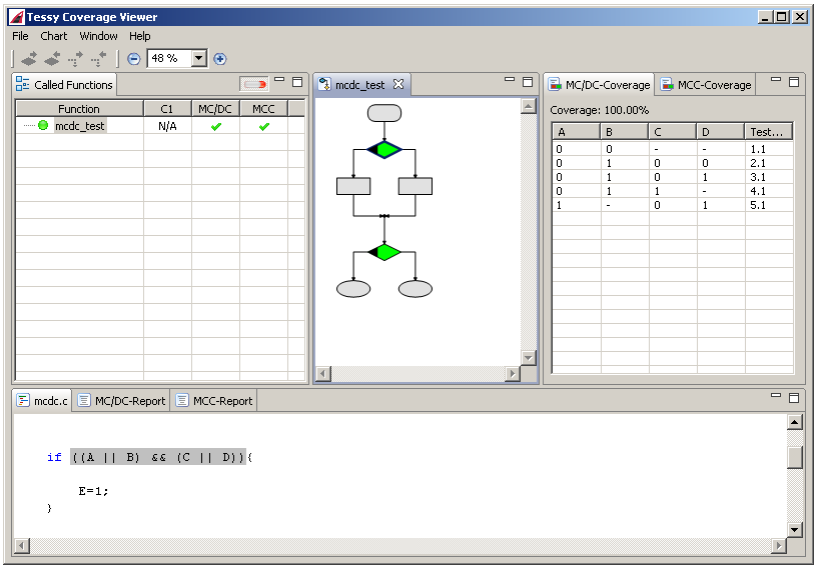
Branch Coverage for selected Test Steps

If you select individual test cases or test steps within the **Testcase / Teststep** list, the respective branches covered by those test steps will be highlighted in green within the flow chart. This allows finding out the execution path of the selected test step. By selecting multiple test steps, you may review the resulting cumulated branch coverage within the flow chart. The **Total Coverage** number will also be updated with the C1 branch coverage for the selected test cases / test steps.

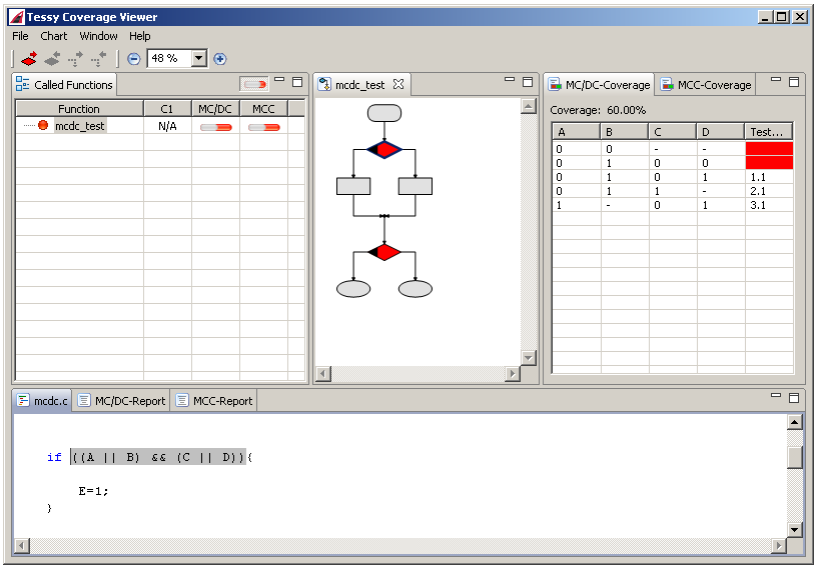
MC/DC Coverage View

The **MC/DC-Coverage** view displays the coverage of the currently selected decision within the flow chart view. If no decision is selected (as initially when starting the CV), the MC/DC-Coverage view is empty.

When selecting a decision, the respective combination table according to the MC/DC coverage definition will be displayed within the **MC/DC-Coverage** view like shown below. The example decision consists of 4 atoms. All condition combinations are fully covered by the sample test steps.

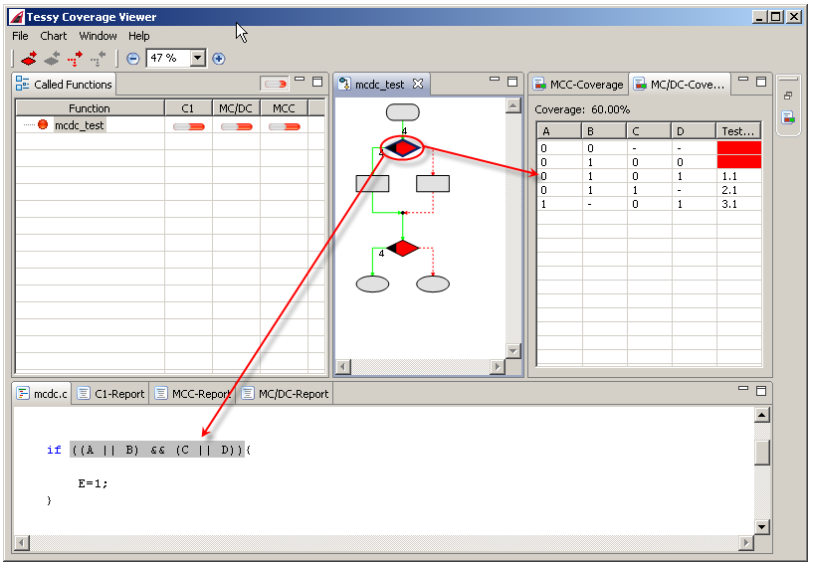


The combination table contains all atomic conditions of the decision. The conditions are the basic atoms of the decision which remain after removing the **or (‘||’)**, **and (‘&&’)** and **not (‘!’)** operators from the decision. Tessy calculates the MC/DC set of true/false combinations of the condition atoms that fits best to the test steps executed during the test run.
The last table column contains the test step that caused the execution of the decision with the true/false combination of the respective table row.
If one or more of the condition combinations were not reached during the test run, the **Teststep** column of those rows will be marked in red like shown below:

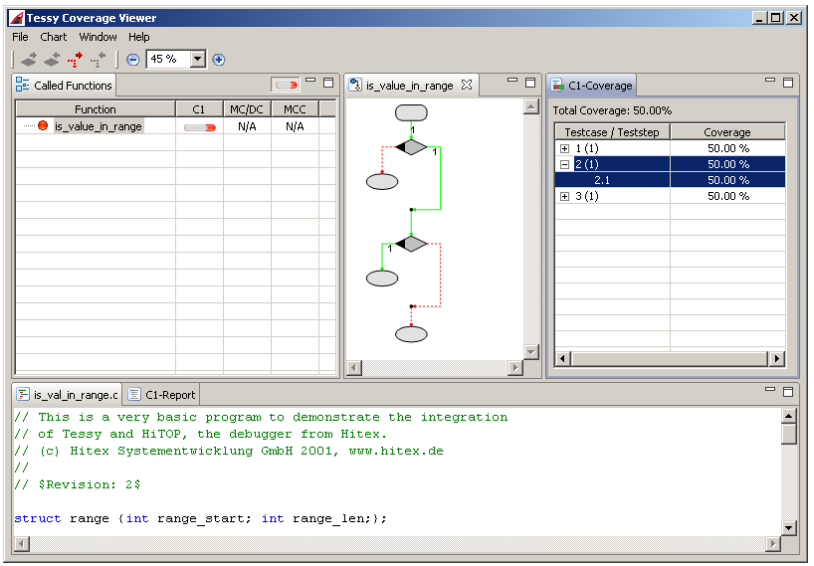


Selecting Decisions

You may select every decision of the program by simply clicking on the respective control flow element within the flow chart view. The respective code fragment will de highlighted within the source code view.



The decisions are either green or red depending on the degree of coverage. If no C2 coverage information is available (e.g. when you ran the test without C2 instrumentation selected), the decisions within the flow chart will appear in grey and the **MC/DC-Coverage** view will not be available (N/A).



Uncovered Condition Combinations

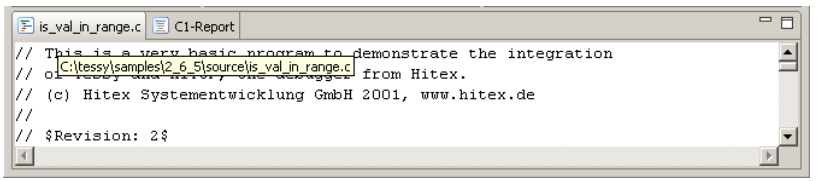
In case of uncovered condition combinations within a decision, you may easily find out the respective settings for the atoms within the combination table. By analyzing the code of the decision, you may use the true/false settings of the combination table within another test step in order to cover the missing combination. You should pay attention to conditions that are negated, because this may be confusing when analyzing the combination table settings.
When adding test steps with other (up to now uncovered) condition combination settings, you may find out that (as a result of a new test run) the order and content of the **Teststep** column have changed. This is the result of Tessy's calculation to find out the best fitting set of test steps for the given decision.

MCC Coverage View

Refer to the description of the MC/DC Coverage View. The only difference is the combination table of the decisions as they are calculated according to the definition of the MCC coverage.

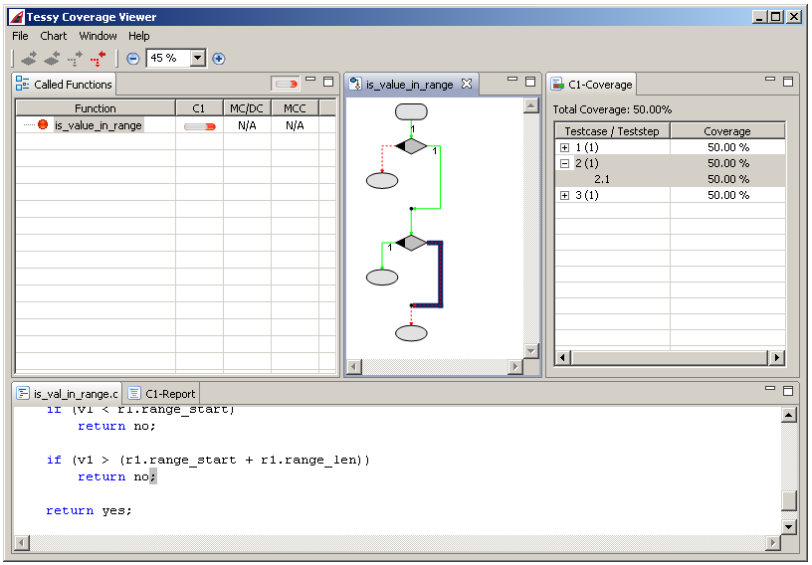
The Code View

Depending on the currently selected function within the **Called Functions** view, the respective source file will be displayed within the code view. There is a tooltip available when moving over the source file name (i.e. the name of the code view) displaying the full path name of the source file.



Highlighting Code Fragments

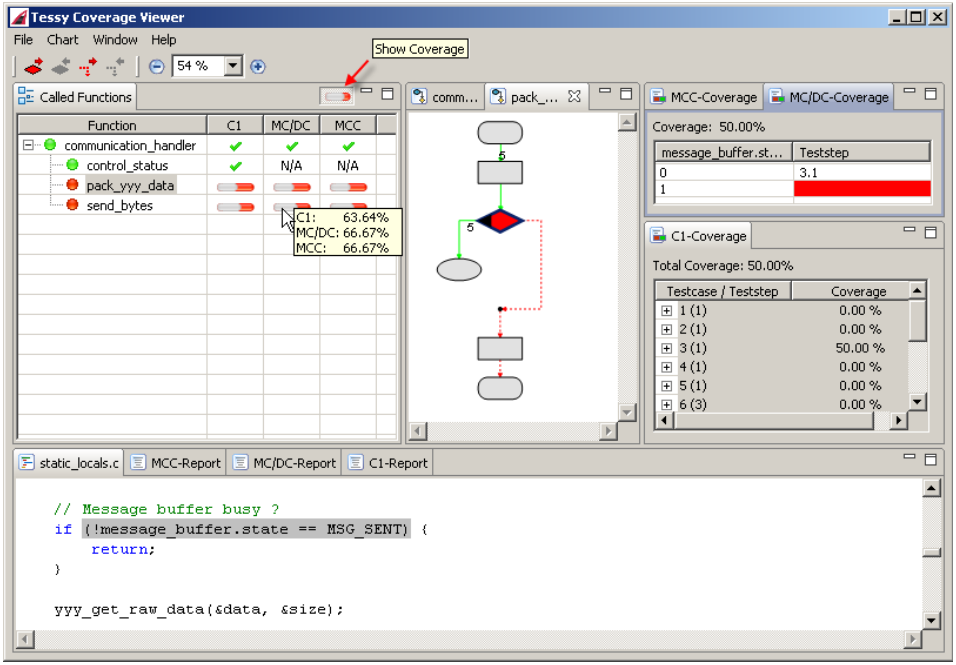
When selecting elements of the flow chart view, the respective source code lines will be highlighted within the source code view. All elements of the control flow and most of the connection lines (i.e. those representing C1 branches) are selectable.
Omitted code statements (e.g. the missing else branch of an if statement) will be visualized by highlighting the last character of the previous statement where the omitted one would be placed.



The picture above highlights the missing else branch of the if statement.

Coverage Statistic View

CV displays the summary of the coverage for C1, MCC and MC/DC in the **Called Functions tab**. You may activate the view with the **Show Coverage** toggle button.



A green tick in the **C1**, **MC/DC** or **MCC** column indicates that 100 % coverage has been reached. **N/A** indicates that the respective coverage measurement is not available for the function. The gray/red bar displays the currently reached coverage for the respective coverage measurement, scaled in 10 % steps. You can move the mouse pointer over the bar to display the values as tool tip.

Coverage Report Views

There are up to three coverage reports available depending on the instrumentation mode selected for test execution. They contain the summarized coverage information of the last test execution:

- The C1 coverage report contains some meta information (e.g. number of branches, reached branches, total branch coverage) and the source code of the test object (as well as of all called functions, if applicable) with the branch counters added in a separate column on the left side of the document.
- The MC/DC coverage report lists all decisions of the test object code (as well as of all called functions, if applicable) including the coverage tables with the respective MC/DC condition combinations.
- MCC coverage report also lists all decisions of the test object code (as well as of all called functions, if applicable) including the coverage tables with the respective MCC condition combinations.

These report documents are already available as ASCII text files within the module directory of the respective test object after the test run. The report files are named like follows:

```
ts_c1_<test object name>_report.txt
ts_c2_mcc_<test object name>_report.txt
ts_c2_mcdc_<test object name>_report.txt
```

The CV displays these reports for convenience and provides a save operation to store a report as text file within another location than the default module directory.

Regression Testing

Performing Regression Testing

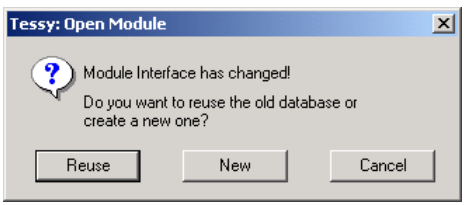
Source files are often modified extensively during the later stages of development, thus making it necessary for a new test to take place, i.e. a regression test. This can alter the interface of an existing test object.

In order to allow you to reuse test data that has already been entered, the **Interface Data Assign Editor IDA** allows the assignment of old interface elements to new ones. This way, old test data can be reused automatically.

The Interface Data Assign Editor IDA

It's not possible to directly start the **IDA**. Whenever a module is opened, Tessy checks, if any modifications to the source file have been made.

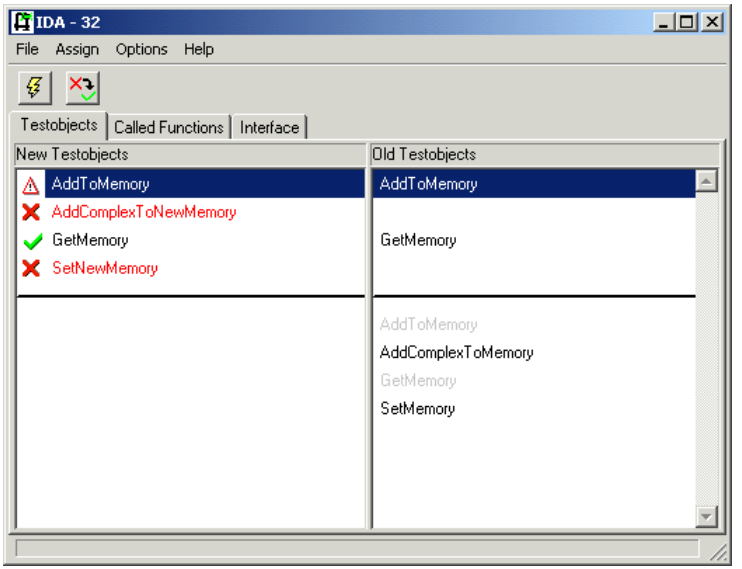
In case of changes, Tessy will open following dialog:



Choose either **Reuse** to keep the old interface information and test data for further processing or choose **New** to delete all settings to start from scratch.

Warning! *Again, clicking on* ***New*** *will remove all test data and interface information from the test database!*

- To keep all test data, click on **Reuse**. *The IDA will open.*



Note: *It is recommended to save the module before using IDA. It might be possible to corrupt the test database due to erroneous assignment of functions (see Save a Module).*

The appearance of **IDA**, i.e. the presence of additional **Tabs**, depending on the test objects used. Normally, **IDA** will start at least with two tabs: **Test Objects** and **Interface**.

IDA will provide additional Tabs for **Called Functions** or **Enums**, if the source file contains functions called from the test object, or enums.

Assigning Test Objects

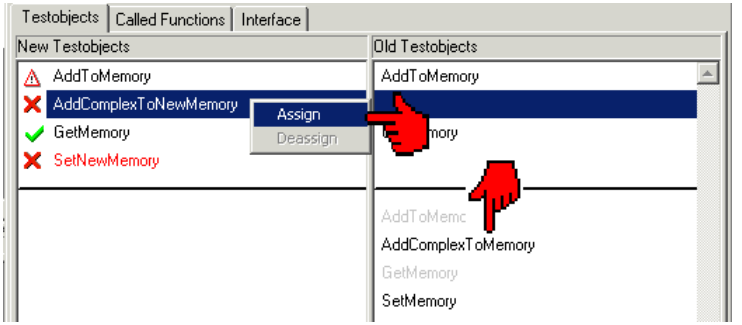
The window of the **Test Object Tab** is vertically split into two areas:

- the left pane contains all New Test Objects
- the right pane contains the Old Test Objects, i.e. all previously existing functions.

  Old functions that have not yet been automatically assigned to new ones are shown under the black horizontal line.

Every new function that is currently available can be assigned to an old function as follows:

- **Use Drag-and-Drop**:
  Use Drag-and-Drop to assign a new function to the desired old function (you can also drag an old function to a new one).
- **Use Context Menu**:
  Select a new function and choose **Assign** from the context. Click the desired old function (or vice versa).



In both cases, the mouse pointer will change to indicate your assignment.

All functions can be assigned automatically as follows:

- Either click the **Automatic Assign** symbol in the toolbar or choose **Automatic Assign** from the **Assign** menu.

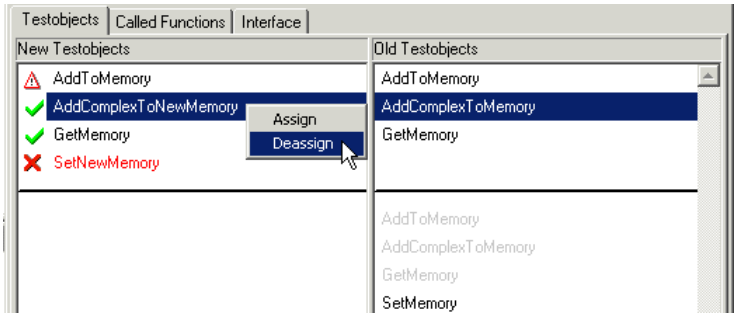
  IDA will try to assign all functions using names.

When an old function has been assigned to a new one, its name will appear next to the new function's name, i.e. the function will move up above the black line. The appearance of the old function under the horizontal black line will change to gray.

Assignments already made can be replaced with new assignments. It's also possible to assign one old function to several new functions.

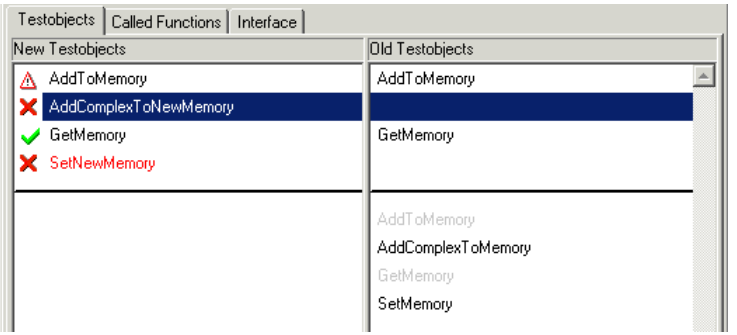
Undoing Assignments

- To undo an assignment, select the respective function and choose Deassign from the context menu. *IDA will deassign the function.*



Symbols

IDA tries to assign all elements automatically. Functions with names that have been changed cannot be assigned automatically. They must be assigned manually using Drag-and-Drop.



IDA will mark all functions to indicate the respective status. Following symbols are used:

	Function is assigned, interface is identical.
	Function is assigned but interface has changed. You need to assign the interface of the test object. • Double click the function or select the function and choose the Interface Tab to assign the interface.
	Function could not be assigned. • Try to assign the function manually (if any).

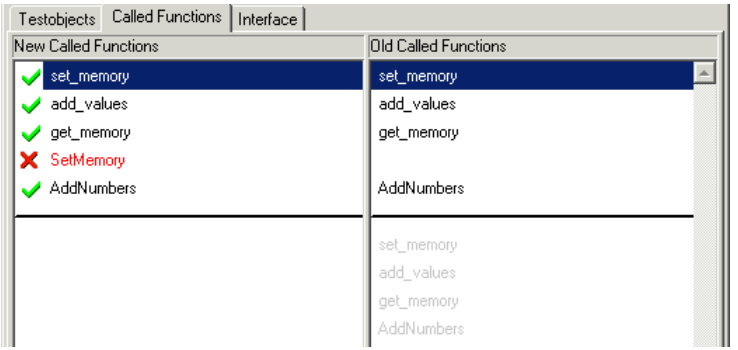
- Use **Sort** from the **Options** menu or press `[F5]` to sort the function list.

Assigning Called Functions

The mechanism to assign called function is the same as for test objects described above. There is one important point to keep in mind when using advanced stubs or get/set functions within your test objects:

- If the name of the called function has changed, then you need to assign the called functions first, before assigning the interface of your test object.

  As a result, you will see the synthetic advanced stub variables within your test object's new interface.

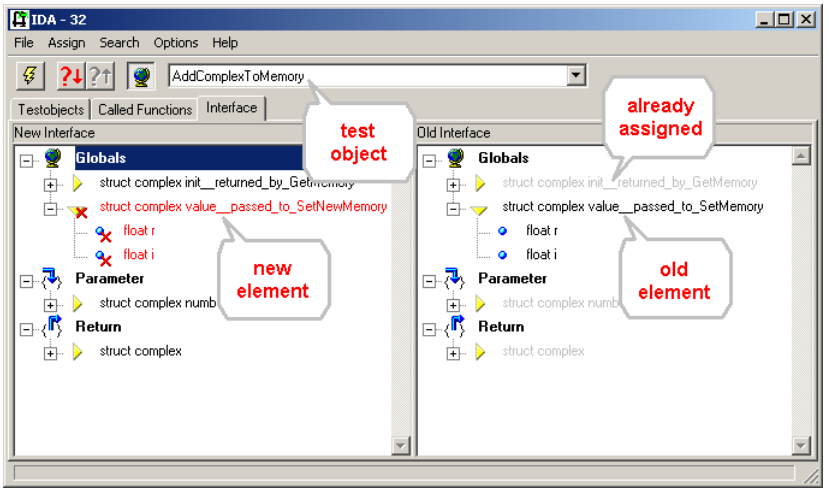


Note: *Please refer to the application notes Using Stub Functions from the* ***Help|Documents*** *menu for details about advanced stub usage.*

Assigning Interfaces

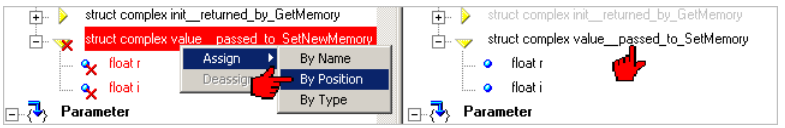
The **Interface Tab** displays the interface of the test object just it would appear in the **TIE**. All new interface elements are shown in the left pane of the window.

IDA will mark each interface element by a red cross to indicate that no automatic assignment took place. Old interface elements which has been already assigned will appear in gray.

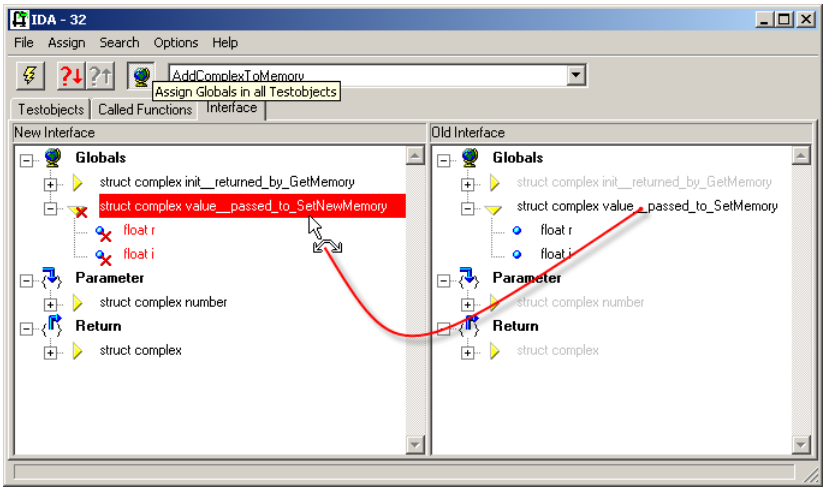


To assign interface elements:

- Select the new interface element and choose **Assign|By Position** from the context menu (for instance).



- or simply click the respective old element and drag it to the new element (or vice versa).



***Attention**: When structures are assigned using Drag-and-Drop, components are assigned according to settings in the **Options** menu (s. Specify Options for Automatic Assignment). If you are unsure of these settings, recheck them once more.*

IDA will assign all global variables to all test objects by default. You may disable this option by clicking the respective icon in the toolbar.

***Note:** You can also assign a single old element to several new elements.*

Jump to undefined values:

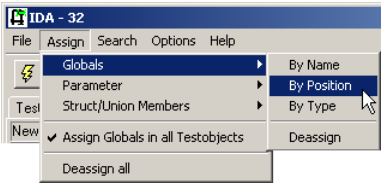
The **Next Undefined Value** and **Previous undefined Value** buttons can be used to jump to the next or previous new interface element that has to be assigned.

Both buttons become inactive as soon as all interface elements are assigned.

Automatic assignment:

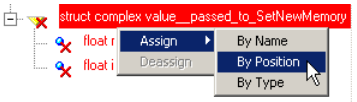
Automatic assignment can also be performed on certain groups or kinds of interface elements.

- Choose an appropriate sub-menu from the **Assign** menu:



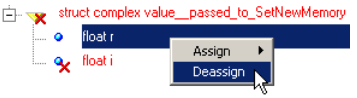
Note: *The option "Struct/Union Members" of the Assign menu can be applied only to already assigned variables.*

All options of the **Assign** menu are also available from the context menu on a selected interface element.

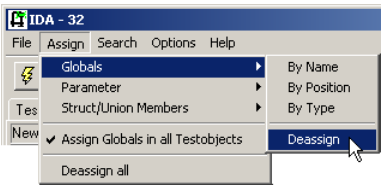


Undo Assignments:

- To undo an assignment, select the respective interface element and choose Deassign from the context menu.



- You may also use the **Deassign all** option from the **Assign** menu to undo all interface elements, or choose only one kind of interface element from the respective sub-menu (e.g. Globals).

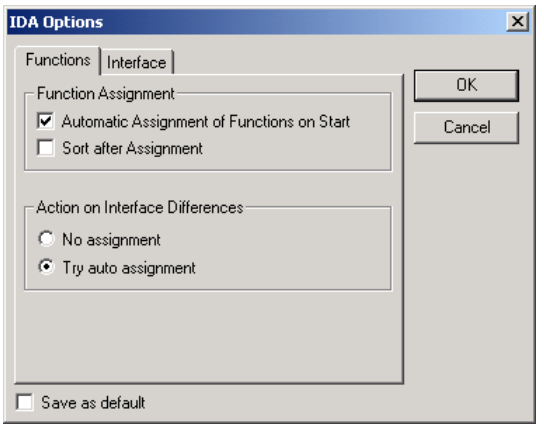


Specify Options for Automatic Assignment

The options menu allows you to set options for the automatic assignment of functions and interface elements.

Functions Tab

The following options are available from the **Functions** tab:



Function Assignment Panel:

- **Automatic Assignment of Functions on Start**
  IDA will attempt to perform an automatic assignment of functions and interface elements whenever started.
- **Sort after Assignment**
  The list of functions will be automatically sorted after each assignment.

Action on Interface Differences Panel:

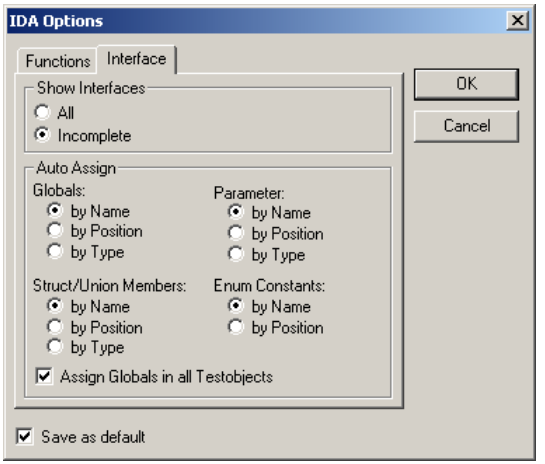
One of the following two options can be selected for the interface assignment of an individual function:

- **No assignment**
  IDA will not assign any elements. Each element will mark by a warning sign to indicate that.

- **Try auto assignment**
  IDA attempts to assign interface elements according to settings within the Interface Tab.

Interface Tab

The following options are available from the **Interface** tab:



Show Interface Panel:

With these options you can decide to keep all interface elements visible within the Interface window after completion of the assignment.

- **All**
  All interface objects will be visible.
- **Incomplete**
  All interface objects will be disappeared (on tab changing).

Auto Assign Panel

These options allow you to set the rules for the automatic assignment of interface elements.

Click **OK** to accept the settings made. Ensure that the check box **Save as Default** is selected, so that the new settings will remain effective for subsequent starts of the **IDA**.

Accepting Changes and Exiting IDA

To commit your assignment:

Click the **Commit Assignment** symbol or use `[Ctrl]+[E]`to accept settings for the new interface structure.

Traceability of Test Changes

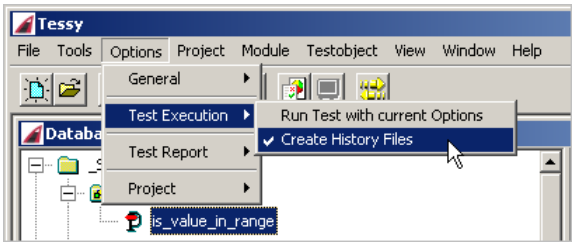
Diff Viewer

Tessy provides a so called Diff Viewer tool which allows to visualize the differences between two versions of the same module test (i.e. before and after a reuse operation). Test versions are stored within XML files and snapshots may be taken on user demand as well as automatically (i.e. before each reuse operation).

The tool may also visualize the differences between two XML report files, enabling later analysis of changes between the tests of two software versions. The tool shows the differences in input and expected values as well as within usercode (i.e. prolog and epilog sections).

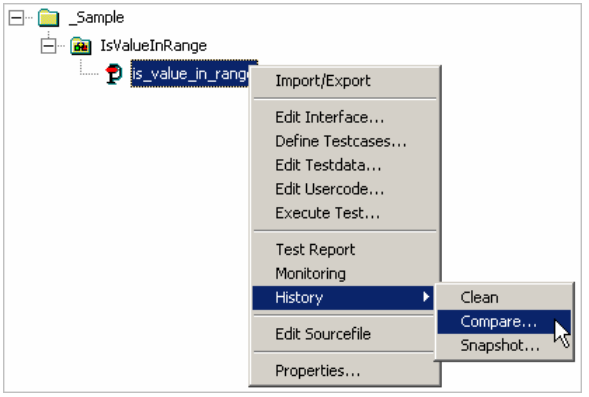
Automated Collecting of Test Results

By default Tessy records automatically the test result of each test execution of a test object. Every test result will be stored in the `.history` folder of the module in a separate `.xml` file. You may disable or enable this feature by using **Create History Files** from the **Options|Test Execution** menu.

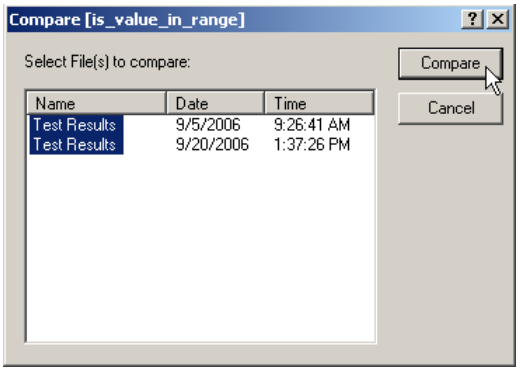


Compare Test Results

To compare test results, please choose **History|Compare…** from the test object context menu.



Tessy lists all **Test Results** with their date and time in the **Compare** dialog.

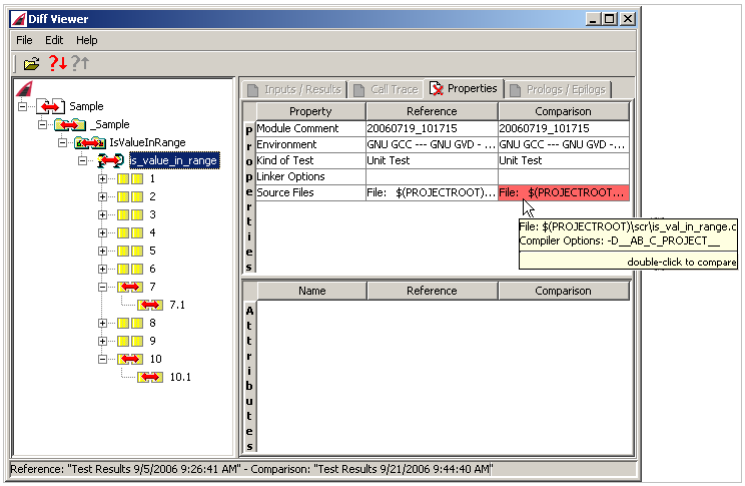


- Select one **Test Results** (compares the actual test results with a selected test results) or two **Test Results** (compares both selected test results).
- Click **Compare**. *The Diff Viewer will appear.*

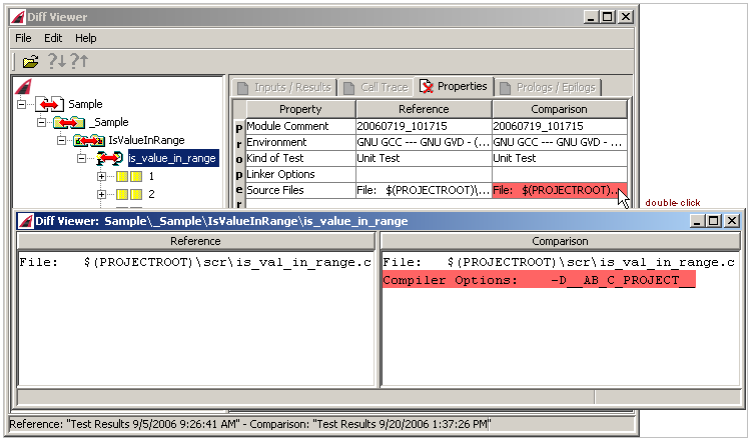
The Diff Viewer is divided into three areas.

- The left pane shows the project database name, the project folder and the module of the test object and the respective test cases.
- The upper and lower panes on the right side display the respective differences of an item as soon as it was selected. The kind of information depends on the tabs (Inputs/Results, Call Trace, Properties, etc.) which have been selected. Only suitable tabs will be enabled for a selected item. In addition, the respective tab will be marked with a red cross to indicate differences.

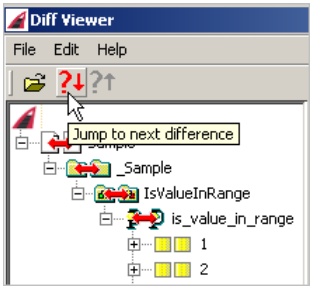
The Diff Viewer highlights every difference in red. You may move the mouse pointer over the red marked field to get a tool tip with the differences.



You may also double-click the field to get another window that shows you the differences.



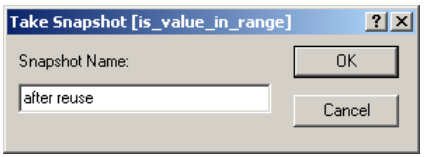
You may use the **Jump to next difference** button to navigate through the project tree.



Create Snapshots

At any point in time you may create a **Snapshot** of a test result. This may be suitable if you have disabled the automatic feature of Tessy (s. Automated Collecting of Test Results).

- Select a test object and choose **History|Snapshot…** from the context menu.



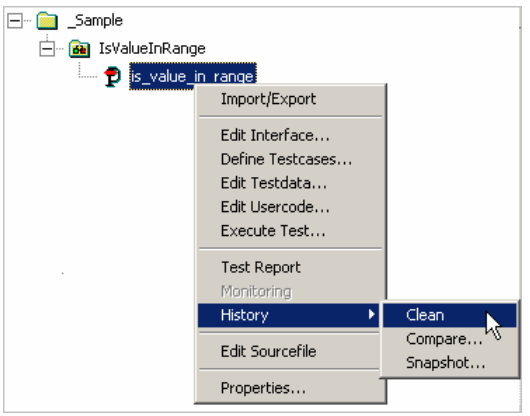
- Enter a name and click **OK**.

Clean History

Every test result will be stored in the `.history` folder of the module in a separate `.xml` file. You may remove the history of test results if you have finished your tests.

To clean the history:

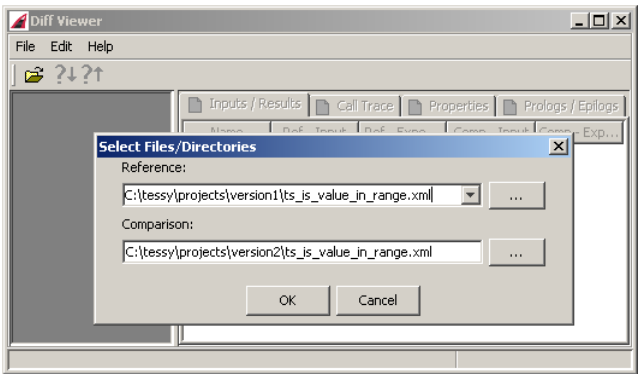
- Select the respective test object and choose **History|Clean.**



Compare XML Reports

You may also use the Diff Viewer tool to compare two XML report files of the same test object.

- Choose **Programs|Tessy|Diff Viewer** from the windows Start menu.
  The Diff Viewer will appear.



- Choose **File|Open** to select two XML Report files (Reference and Comparison) of the same test object.
- Click **OK**.

Troubleshooting

Contact Support

In this chapter we offer solutions to some commonly asked questions and error messages that may occur using Tessy. If you have further questions as well as error reports, please contact our technical support.

Phone:	+49 (0) 30 53 63 57 0
Fax:	+49 (0) 30 53 63 57 60
e-Mail:	support@razorcat.com
Internet:	http://www.razorcat.com

Enable Logging for Troubleshooting

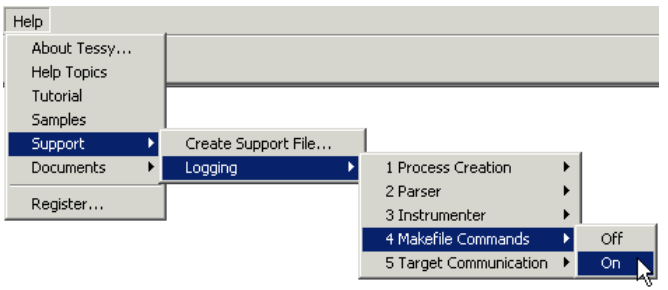
In case of problems, you may enable the logging feature of Tessy for troubleshooting. Enabling logging will create more information within the message window of Tessy.

There are several debug level available, that will cover each kind of steps during your testing

- Process Creation
- Parser
- Instrumenter
- Makefile Commands
- Target Communication

Please do the following to enable logging:

- Start Tessy.
  (please close TIE, TDE, Usercode editor, if open).
- Choose **Help|Support|Logging…** and then an appropriate command e.g. **Makefile Commands {On|Off}** to enable or disable logging.



- Clear the message window by choosing **Delete all** from the context menu.
- Repeat test execution with **Generate Driver** and **Run**.
- Analyze the message window for troubleshooting.

Open Support Dialog

In case of problems during compile/link or test execution, Tessy can create a file containing all information to help Razorcat Support for troubleshooting.

Please do the following:

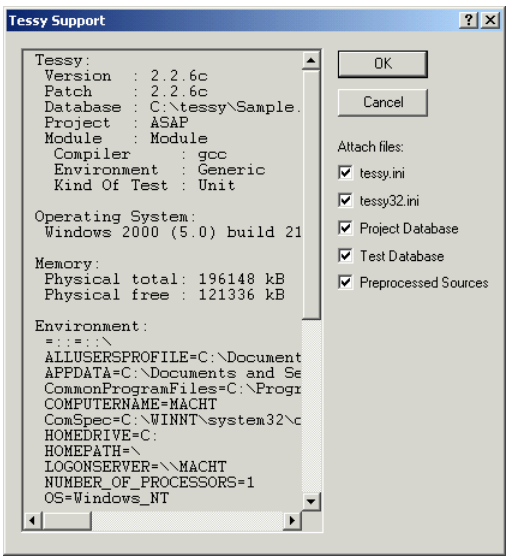
- Enable logging. This may be helpful for troubleshooting (s. Enable Logging for Troubleshooting).

Problem: The Module cannot be opened:

- Clear the message window by choosing **Delete all** from the context menu.
- Choose "**Check Sourcefile(s)**" from the module context menu. Try to open the module. Continue with step 4.

Problem: The next steps are suitable if the module is open but the test execution failed.

- Clear the message window by choosing **Delete all** from the context menu.
- Select your test object within the module that caused the problem.
- Repeat test execution with **Generate Driver** (and **Run)**.
- Choose **Support|Create Support File…** from the **Help** menu. *Tessy Support* dialog will appear.



- Tick all necessary information and click **OK**. The next dialog shows the path where the support file **TessySupportInfo.tar** has been created (default: `c:\Tessy`).

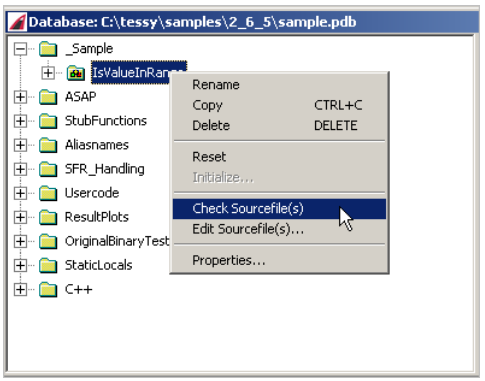
***Note**: You should also tick the Preprocessed Sources if possible. This enable us to execute the test object with Tessy.*

- Please send **TessySupportInfo.tar** to support@razorcat.com.

Error Messages

Error messages printed when opening a module or during the test run are caused by errors during preprocessing or compilation of the source code. Mostly, these are syntax errors in the source file.

Please check your source files using the **Check Sourcefile(s)** command from the context menu of the module. This will use directly your cross-compiler, e.g. Tasking C166. The error messages of your compiler will be printed into the Tessy message window.



How to Handle Errors Appearing in the Message Window

The most relevant error will be found at the top of the error output. The subsequent errors are mostly due to the previous ones and may be confusing. We recommend to follow the two steps described below, when searching for an error reason:

- At first, delete the content of the message window.
- Repeat the erroneous action.

Please mind! The error messages of your compiler will be printed into the Tessy message window. In case of syntactical errors or linker errors, please refer to the manuals of your compiler.

Common Error Messages

No such file or directory

Solution: Please find out, which file exactly was not found. Most commonly, the compiler fails to open some include files. In this case, you have to add the include paths in the Defines tab of the module properties.

Undefined reference to ...

Solution: Please check the settings of stub functions and external variables within TIE. Probably you need to define the missing external variables or stub functions.

Multiple definition of ...

Solution: Please check, whether external variables or stub functions are defined more than once at different locations, e.g. in the TIE as well as in other function libraries.

Please also check the Usercode, whether a variable was defined under **Declarations**. Variable definitions *must* be entered into the **Definitions** section, because the code of Declarations will be inserted into different header files during test driver generation.

Tessy Error Codes

The following table lists error codes of the interface analyzer and error codes during the test preparation and test execution. In case of internal errors, please contact technical support.

***Note**: In case of errors, please check also the message window for further information. The possible reason will be displayed before the error code.*

Unit	Error code	Error description

Unit	Error code	Error description
TDB	1 - 42	Internal errors
File IO	30	Filename too long
	31	File not found
	32	Already open
	33	Not open
	34	Open error
	35	Write error
	36	Read error
	37	Close error
	38	Delete error
Analyzer	50	Syntax error
	51	Stack overflow
	52	Semantic error
	53	No public function
	54	No name space
	55	Parse error
	56	Static name too long
IDB	101 - 111	Internal errors
	150	Preprocessed source file not found, probably due to preprocessing errors. *Use Check Sourcefile(s) from the context menu of the module to use the cross compiler.*
	151	Preprocessed source name too long.
	152 - 154	Error reading preprocessed source file. *Check the permissions of the module directory and the source file.*

Unit	Error code	Error description
	170 - 177	Internal errors
	200	Bad type definition
	201	Multiple declaration of tag names
	202	Bad usage of tag names
	203	Unknown tag name
	204	Multiple declaration of type names
	205	Could not calculate bitfield size. *You need to enter the size manually within the TIE.*
	206	Bad type for bitfield. *Check the compiler settings (e.g. the gcc allows only signed and unsigned for bitfields).*
	207	Multiple declaration of symbol
	208	Bad initializer for array dimension
	209	Error in function definition. *Check the compiler settings: This indicates an error with a certain compiler specific keyword. Contact suppport, if Check Sourcefile(s) command from the context menu of the module succeeds.*
	210	Multiple definition of function
	211	Could not calculate array dimension(s). *You need to enter the dimension size manually within TIE.*
	212	Could not calculate enum constant. *You need to enter the value of the enum constant manually within TIE.*
	300	Syntax error in source file. *Use the Check Sourcefile(s) command from the context menu of the module.*
	301-303	Internal errors

Unit	Error code	Error description
Test Preparation	661	Error during test preparation. *The possible reason will be displayed before the error code. The reason is dependent of the target used.*
Test Execution	666-667	Error during test execution. *The possible reason will be displayed before the error code. The reason is dependent of the target used.* *You may debug your test object. Please use option "Define Breakpoint" in the Execute Test dialog for this purpose.*

FAQ

Errors While Opening a Module

Problem: Some error messages show up when opening a module:

Solution: Such problems are either caused by erroneous compiler options or syntactical errors in the source file.

- Use the **Check Sourcefile(s)** command from the context menu of the module to make sure, that your source files compile correctly.
- Check the messages in the message window and correct your source files if necessary.
- Did you choose the right compiler?
  The GCC (Gnu C Compiler) doesn't understand certain target compiler types, e.g *bit* and *bitword.*
- Is the source file available?
  Check within the module properties dialog, whether the source files are really located in the specified directory.
- Please keep in mind, that at least *one* function in the source file(s) must not be declared as "`static`" even if you check the **Enable Static Functions** option within the module properties.

Errors While Generating the Test Driver

Problem: Some error messages show up during test driver compilation and linking.

Solution: These problems are either caused by errors in the source files or within the usercode, erroneous TIE settings or erroneous linker options.

- Check the messages in the message window and correct your source files if necessary.
- Please notice: The error messages of your compiler will be printed into the Tessy message window; In case of syntactical errors or linker errors, please refer to the manuals of your compiler.
- Check your inputs to the usercode and the stub functions.
- Please check the settings of stub functions and external variables within TIE. Probably you need to define the missing external variables or stub functions.

Module Interface has Changed

Problem: A dialog with the message "`Module Interface has changed!`" appears while opening a module.

Solution: Tessy has detected, that the interface of the test object has changed. You can either click **Reuse** in order to start the Interface Data Assigner **IDA** for reusing your test data. If you want to delete all test data and restart from scratch, click **New**. All test data will be deleted in this case.

Test Cases Can't be Inserted

Problem: The **Insert** command from the context menu of the test case list is not available (on the right side of the Tessy main window).

Normally, Tessy tries to analyze all interface objects while opening the module. If an "unresolved interface object" has been found, Tessy will not open the test object clamp to indicate that. In that case you cannot create a test case.

Solution: In order to create a new test case, the interface data of the test object has to be checked or reviewed first within TIE. Please open the TIE and check all unresolved interface objects.

To open the TIE, click the right mouse button on the desired test object and choose **Edit Interface** from the context menu.

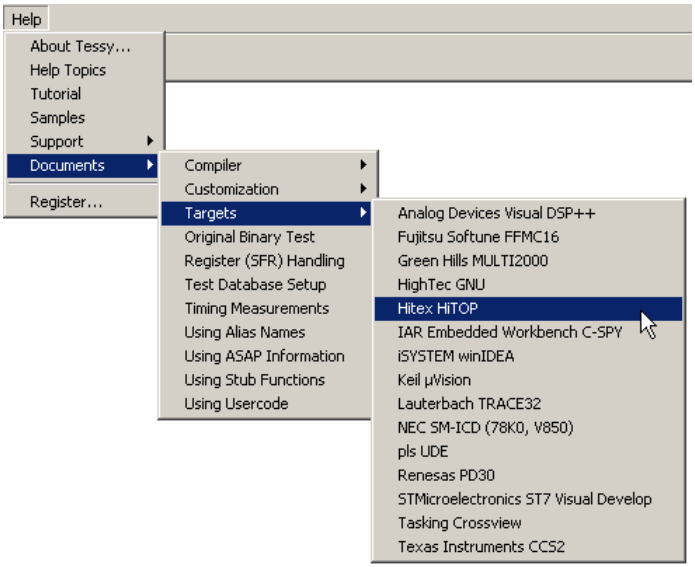
I Can't Choose a Certain Compiler or Debugger

Problem: The desired compiler or debugger/emulator is not available in the list within the module properties dialog.

Solution: You need to open the so called **Tessy Environment Editor** to enable your target environment.

Application Notes of Tessy

All available application notes have been copied to the documentation folder of Tessy where the tool has been installed. For your convenient, you may open these documents directly from the **Help|Documents** menu of Tessy.



You will find documents both for common topics, e.g. “Using Usercode”, and target specific topics, e.g. “Hitex HiTOP”.

Technical Information

Overview

The target compiler and debugger/emulator integrations will be described in the following sections. Later on, some more technical details of Tessy will be explained, that will enable the experienced user to modify the Tessy installation. These changes should be carried out very carefully, since erroneous changes may result in severe failures of Tessy.

Our technical support will be pleased to assist you in any adoption to your specific project environment and implementation of required customizations.

Compiler Settings

Tessy supports different target compilers for use with emulators or debuggers on the target platform. Tessy supports both host-based testing (using the GNU gcc compiler for Windows) and cross compilation for various platforms. Please review our web page for the actual list of supported compilers and microcontrollers:

http://www.razorcat.com

For each compiler and target system combination, there is a specific makefile template installed that may be adapted to your needs. When using the standard installation you just need to review the default settings.

Tessy uses a so called `typetable.xml` (`$(TESSY_SYSPATH)\compiler`) to determine the various data types supported by each compiler. In some cases you need to adapt the default settings for your used compiler, e.g., `enum_size, enum_sign.`

If you want to use some specific compiler options or specify default include paths for your projects, then you need either to make these changes in the corresponding makefile template or you have to use the Environment Editor.

Please refer to the application notes "Customizing Makefile Template" from the **Help|Documents** menu for more details about how to do such changes.

Please Note: *Certain restrictions apply to the usage of registers. Please refer to the application notes "Register (SFR) Handling" from the* ***Help|Documents*** *menu for tips and hints on how to solve common problems.*

Debugger/Emulator Settings

Tessy supports different target emulators, debuggers or simulators. Especially the emulators require some basic setup concerning memory mapping and default settings for the communication link between Tessy and the respective target system. Please review our web page for the actual list of supported compilers and microcontrollers:

http://www.razorcat.com

Refer to the application notes for tips and hints on common problems. You will find the respective application notes for every target debugger or emulator within the **Help|Documents|Targets** menu.

Please note: *The Original Binary Test using emulator scripting features is only available for selected targets. Please review the application note "Original Binary Test" from the* ***Help|Documents*** *menu for details on the supported target platforms.*

Version Control

If you want to version your tests together with your source code, you need to archive files from the module folder and other files which configure the test environment.

When restoring files from the version control system, make sure that you remove the write protection of the files. Otherwise, you will not be able to open the modules.

Hint: *If you want to move the whole database directory to another location, please refer to applications notes "Test Database Setup" from the* ***Help/Documents*** *menu.*

Files from the Module Folder

There are two possibilities to store Tessy files for version control:

- You may store the database root folder tree containing the project and module directories to save the necessary files: `*.pdb, *.tdb, *.idb, *.tpl, *.cte` (you don't need to store `ts_new_*.idb`). You should be able to restore these files in the given directory structure.

`<project database name>.pdb`	Project database, contains information about projects and modules.
`<module name>.idb`	Interface database, contains the module interface information.
`<module name>.tdb`	Test database, contains your test data.
`<test object name>\*.tpl`	Template files, contains Usercode and code of stub functions.
`<test object name>\.cte`	The CTE files you created using CTE.

- You may create module backup files for each module and store these *.tmb files within the version control system.

The tmb file is simply a tar archive which contains all necessary files of one module. You can create this file by using **Module|Save** or **Module|Restore** to import the module back in a Tessy project folder (s. Save or Restore a Module).

Files outside the Module Folder

The Tessy Environment Editor specifies all settings which you have done for your target environment. Which files must be stored depends on the used target, some are:

- Makefile Template
- Linker File
- Project Files for the debugger
- Libraries

You may copy such files in a separate folder other than Tessy's default folders to store these files also in the version control system. You may create a specific configuration file for a given project database (s. Creating Configuration Files) to store your test environment settings in the version control system too. Tessy will recommend a **`config`** folder within the project root by default if you create a new project database.

Files in the Module Directory

The module directory contains all files necessary for the test run. Beside the files containing user data (Refer to "Version Control" above), there are a lot of generated files.

Makefile

For each test object, there will be a makefile created for compilation and linking of the test driver files (**`ts_<testobject name>.mak`**). This makefile is based on a makefile template. If you need to change the makefile, you need to do the changes in the corresponding makefile template (refer to "Makefile Templates" below).

Some compilers require additional files (e.g. linker command file), that will be created through rules within the makefile. For changes on these generated files, you also need to change the makefile template.

Makefile Templates

The makefile templates are located in the directory “sys\templates\make” within the Tessy installation directory. The name of the makefiles depends on the corresponding debugger/emulator, compiler and microcontroller to be used. The following naming convention applies:

```
ts_make_<debugger>_<compiler>_<processor>.tpl
```

Example:

```
ts_make_hitop_keil_c166.tpl
```

You may change the makefile templates in the following designated sections:

```
#BEGIN_DEFINES
Here you can add defines or change existing ones.
#END_DEFINES

#BEGIN_TARGETS
Here you can add rules or change existing ones.
#END_TARGETS
```

Please refer to the application notes “**Makefile Templates**” from the **Help|Documents** menu for more details about how to do such changes.

Generated Files

The following generated files contain code that has been entered by the user (source code, Usercode, stub functions). In case of errors in the code provided, these file names will be shown in the error messages during compilation and linking.

ts_<testobject name>_usr.c	Contains Usercode (prolog/epilog/definitions)
ts_<testobject name>_stubs.c	Contains code of stub functions
ts_<testobject name>_s.c	Contains the test driver (with declarations/definitions of variables used in your code)

Technical Restrictions

The following restrictions and limitations apply for Tessy.

Number of Test Cases and Test Steps

The number of test cases is restricted to 995 and the number of test steps for each test case is restricted to 999.

Depending on the size of the test object interface, you will achieve satisfactory performance with up to 300 test cases.

Tasking Compiler bit Variables

Due to the reason, that the Tasking compiler doesn't allow indirect access to bit variables (via pointer to bit), bit variables cannot be assigned or evaluated, if they are used within `structs` or `unions`. You need to set the passing direction of those `structs` or `unions` to `Irrelevant`.

There is also a restriction for static local bit variables defined within a function or statement block. Tessy may only assign input values for those variables because of technical reasons. You need to set the passing direction of those variables to `In` or `Irrelevant`.

Test Object Interface Size

The interface size of your test object should not exceed a certain number (the absolute maximum size is depending on the available resources of your computer). If the size of the interface exceeds the maximum, TIE and TDE will run out of window handles and will be terminated by Windows. In this case, the respective test object cannot be handled by Tessy.

Structs/Unions/Enums and Multiple Source Files

This restriction applies only for Tessy modules with **more than one** source file. Due to the fact, that the interface database of a module holds the combined interface of these several source files, there may appear conflicts for certain type definitions or declarations. In the cases below, it will not be possible to successfully build the test driver:

- Two enums in different source files without tag names or with different tag names or typedef names and at least one identical enum constant that are used within two functions (one function in each of the source files) and one function is calling the other.
- Two structs/unions in different source files with the same tag names or typedef names and differing components that are used within two functions (both functions in different source files) and one function is calling the other.

In both cases, you may nevertheless run the test successfully when testing each source file separately with only **one** source file in the respective module.

Glossary of Terms

Actual Values

Values, which were calculated by the test object during execution. These values are compared with the expected values after the test run.

C1 Test

During the C1 test, each branch of the test object will be instrumented with a counter to monitor, how often a branch of the program is run through.

CTE

Classification Tree Editor

Defined

A new variable is introduced in the source file.

Declared

The name of a variable is declared in a source file. The definition of the variable takes place in another source file or in a library.

Expected Values

These are the values, that are expected to be calculated by the test object. The actual values are checked against the expected values after the test run.

IDA

Interface Data Assign editor: If the interface of the test object has changed, you can assign the new interface to the old interface. Your test data will be assigned automatically.

Input Values

Function parameters, global and external variables which have effect on the behavior of the function.

Interface Description

Information about the passing direction and usage of interface elements (parameter, global variables and function parameters). These settings are made within TIE.

is_line_covered_by_rectangle

The function, which is used in the tutorial. It tests, whether a given line is covered by a given rectangle.

Module

A module contains source files, compiler settings, interface description and test data. You can pool modules in projects.

Output Values

These are the return value, global and external variables, which are changed by the test object. They are evaluated against the expected values after the test run.

Passing Direction

The passing direction indicates, whether a variable effects on the behavior of a function or is changed by it.

- **In:** The variable is read by the test object.
- **Out**: The variable is changed by the test object
- **InOut**: The variable is read and changed as well by the test object.

Project

A project stores modules. It makes sense to store modules, that belong to the same subsystem, into the same project.

TDE

Test Data Editor – With the TDE you can enter the input values and expected values for the test run.

Test Driver

Files generated by Tessy for the test execution.

Test Environment

Information about the test object, the compiler used, the target debugger or emulator and more settings.

Test Object

The function to be tested.

TIE

Test Interface Editor: With the TIE you can view the interface description and set the passing direction of the interface elements.

Usercode

Here you can enter C code, which is executed before or after test cases/test steps during the execution of a test object.

Index

A

AbsRange 248
Add Module 57, 94
Advanced Stub Functions
  create 167
Arrays 230
Assignments
  undo 312
Attributes 111, 115
  CTE File 116
  Execute Testcases Separately 116

B

Batch Test 268
Breakpoint
  define 262

C

Central License Server 42, 43
Check Sourcefile 117, 118
Classes 3
Classification Tree Editor 3
Classification Tree Method 3
Classifications 3
Compiler
  settings 342
Configuration
  typetable.xml 342
Configuration File
  assign 88, 137, 143
  specify 83
Copy&Paste
  modules 95
Coverage Analysis
  C1 261
  MC/DC 261
  MCC 261
Create Database
  project root 82
  root directory 83
Create Target Value 241
CTE
  export to Tessy 183
  open 178

D

Database
  close 90
  delete 90
  root directory 83
  start with last database 90
Database Root 80
Databases
  new 81, 139
  open 85
  root directory 282
Debugger/Emulator
  settings 342
Define Breakpoint 262
Define Testcases 178
Delete

modules 98
test step 224
Description
test case 217

E

Edit Source Files 117
Edit Test Data 216
Edit Testdata 66
Enable checking 234
Enable Static Functions as Testobjects 107
Enter Character 235
Enter Values
CTE exported 230
Error Codes
Tessy 335
Error Messages 331
Eval Macros 200
Evaluate Test 73, 277
Execute Test 71, 259
generate driver 259
run 259
testcases separately 259
Testcases separately 116
Expected Values 69, 152, 229
Export 211
External Functions 154
create stubs 163
External Variables 65, 228

F

File Status 108
Floating License Manager 39
Floating License Server 36
FLS 36
Format
test report 279, 282
Function Parameter 65

G

Generate Driver 264
Generated Files 345
Get/Set Function 171
Global Variables 64, 155, 218, 228

H

Hide Functions 108

I

Icons
colour 253
IDA 309
assign automatically 311
assign called functions 314
assign functions 310
assign interfaces 315
icons 313
interface data assigner 5
regression test 5
save 320
unions 317
IDB Files 96
Import 211
Imprint ii
Init Testdata 232
Input Value
assign value to all test cases 231
Input Values 152, 228
Insert Testcase 66
Install License Server
port number 41
Installation
base directory 14
overview 11
version directory 14
Integration Testing 1
Interface Data Assigner 309

L

Layout
  Test Report 279, 282
License 36
  checking license file 46
  request 39
License key
  request 39
License Key
  Install central 44
  Install local 40
License Server
  central 43
  error connecting to host 47
  local 48
  not a valid license server 47
  run as application 43
  run as service 43
  specify 42
  stop 46
Linker / Binary 109
Local License Server 48

M

Makefile Templates 345
Messages Window 334, 335
  error messages 334
Modify Tessy 341
Module Interface has Changed 339
Module Properties 99, 338
  attributes 111, 115
  comment 114, 117
  general 100
  linker / binary 109
  sources 60, 101
Modules 2, 93
  add 57, 94
  copy 95
  create 57, 94
  delete 98
  paste 95
  rename 94
  restore saved 97
  save 95

N

New
  database 81, 139
  project 91
Next
  undefined ... 68, 230, 316
  unknown object 159
Notebook license 50
notebook_like.dat 50

O

Open Database 85
Open Support Dialog 333
Output Directory
  test report 279, 282
Output Values 69, 152

P

Parameter 65, 155, 219
Passing Direction 64, 157
  bitfields 162
  enumerations 162
  extern 64, 157, 227
  in 64, 157
  `inout` 64, 157
  `irrelevant` 64, 157, 227
  out 64, 157
  pointers 161
  reset 159
  structures and unions 161
`PDB Files` 343
Pointers
  passing directions 161
Prev
  unknown object 159

Project Root 79, 82
  assign 86
Projects 2, 84, 91
  comments 91
  create 91
  delete 93
  open 92
  rename 92
Properties
  module 99
  project 91
Properties tab 56

R

Range 248
Regression Testing 4, 309
Report
  $Author$ 286
  $Date$ 286
  $Revision$ 286
  keyword expansion 286
Return Value 65, 155
Root Directory (database) 282
Run
  test 265

S

Save
  module 95
Search
  identifier 175, 255
  variables 175, 255
Set Pointer
  target 240
Shortcuts
  TDE 255
Show
  all elements 238
  first array element 237
Source File
  changed 118
  changed interface 119
  edit 117
  file status 108
  specify 60, 101
Specification 217
Start Local Server 44, 46
Static Functions 107
Stop Local Server 46
Strong Syntax Checking 234
Structures 236
  passing directions 161
Style
  test report 279, 282
Support
  dialog 333
Systematic Test 4

T

TDB Files 96
TDE
  arrays 230, 237
  bit check 251
  browse Interface 227
  enter expected values 227
  enter input values 227
  enter string 239
  enum variables 236
  start 216
  strings 239
  structures 236
  unions 236
  values for pointers 239
Tessy
  introduction 53
  license 36
  specify source file 60
  start 54
  tutorial 53
Tessy Error Codes 335
Test

evaluate 73, 277
execute 259
Test Case Description 217
Test Case Specification 217
Test Data Editor 4
Test Database 96, 258, 343
Test Driver
Master 258
Slave 258
Test Environment 98
Test Execution
test step repetition 227
Test Interface Editor 151
Test Options 259
Test Report 73, 277, 278, 281, 282
Test Run 257
Test Step
clear test data 225
create 222
delete 224
delete test data 224
Testarea
specify 25
Testcase Determination 1, 3
Testcases 3, 177
delete 187
execute separately 259, 260
insert 66
Testobject 2
properties 115
Testobject Specific Code 209
Testreport 282
Teststep
initialize values 232
TIE 151
bitfields 162
browse interface 154
enumerations 162
external functions 154
global variables 155, 218
parameters 155, 219
passing directions 157
TMB Files 96
TPL Files 96
TS_CALL_COUNT 200
TS_CURRENT_TESTCASE 198
TS_CURRENT_TESTSTEP 198
TS_REPEAT_COUNT 197
TS_TESTOBJECT_RETURN 199
tstcomm interface 258
Tutorial 53

U

Unions 236, 317
passing directions 161
Unit Tab 62
Unit Testing 1
Use local License Server 41
Use remote License Server 42
Usercode 190
enter 193

V

VCS Settings 343
Version Control System
Author 286
Date 286
files to save 343
keyword expansion 286
Revision 286